**Лабораторная работа №4**

**ПОСЛЕДОВАТЕЛЬНОСТИ И КОЛЛЕКЦИИ**

Swift предлагает три фундаментальных типа данных, обеспечивающих функциональность коллекций*: массив (array), набор (set) и словарь (dictionary).* Все три типа коллекций по аналогии с кортежами представляют собой отдельные типы данных.

**Массивы**

Полная форма указания типа массива в языке Swift:

Array<SomeType>

, где SomeType является типом значений, которые могут содержаться в массиве. Вы также можете написать тип массива в сокращенном виде как [SomeType]. Хотя обе формы функционально идентичны, но в сокращенном виде запись, является предпочтительной.

Типом массива является тип-значение (value type), а не тип-ссылка (reference type). Это означает, что при попытке скопировать или передать значение массива создается его копия, с которой и происходит вся дальнейшая работа.

**Создание и инициализация массива**

Массивы хранятся в переменных и константах, поэтому для их объявления используются операторы var и let.

Вы можете создать пустой массив определенного типа, используя синтаксис инициализации:

var someInts = [ Int ]()

print ( "someInts имеет тип [Int] и содержит \( someInts.count ) элементов." )

// prints "someInts элементов [Int] и содержит 0 элементов.»

Если контекст уже обеспечивает информацию о типе, как аргумент функции или типизированная переменная или константа, то вы можете создать пустой массив с пустым литералом массива, который пишется как [] (пустая пара квадратных скобок).

someInts.append ( 3 )

// someInts теперь содержит 1 значение типа Int

someInts = []

// someInts теперь пустой массив, но по-прежнему остается типом [Int]

/\* объявляем массив с пустым значением

с помощью переданного значения \*/

var emptyArray: [String] = []

/\* объявляем массив с пустым значением

с помощью специальной функции \*/

var anotherEmptyArray = [String]()

В результате создаются два пустых массива, emptyArray и anotherEmptyArray, уже инициализированные значениями (хотя и не содержащими элементов)

Тип Array в языке Swift так же имеет свой инициализатор для создания массива определенного размера с присвоением значения по умолчанию всем исходным элементам. Вы передаете инициализатору число элементов, которое должно быть добавлено массиву (count) и значение по умолчанию с подходящим типом (repeatedValue):

var threeDoubles = [ Double ]( count : 3 , repeatedValue : 0.0 )

// threeDoubles имеет тип [Double], и равен [0.0, 0.0, 0.0]

let repeatArray = Array(repeating: "Swift", count: 5)

Вы можете создать новый массив путем сложения двух существующих массивов с совместимыми типами при помощи оператора сложения ( + ). Тип нового массива выводится из типа двух складываемых массивов:

var anotherThreeDoubles = [Double](repeating: 2.5, count: 3)

// anotherThreeDoubles выведен тип [Double], и равен [2.5, 2.5, 2.5]

var sixDoubles = anotherThreeDoubles + anotherThreeDoubles

// sixDoubles выведен тип [Double], и равен [0.0, 0.0, 0.0, 2.5, 2.5, 2.5]

**Литералы массива**

Вы можете инициализировать массив литералом массива, что является сокращенной формой для записи одной или нескольких величин в коллекцию массива. Литерал массива записывается в виде списка значений, разделенных запятыми, в окружении пары квадратных скобок:

[Значение 1, значение 2, значение 3]

Пример ниже создает массив с именем shoppingList для хранения String значений:

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

// shoppingList был инициализирован двумя исходными элементами

shoppingList переменная объявлена как «массив строковых значений», записанная, как [String]. Поскольку для этого массива указан тип хранимых значений как String, то только String значения ему и разрешено хранить.. Здесь shoppingList массив инициализируется с двумя Sting значениями ( «Eggs» и «Milk» ), записанных в  литерал массива .

Благодаря выведению типов Swift, вы не обязаны писать типа массива, если вы инициализируете его литералом, содержащим элементы одного типа. Значит инициализация shoppingList может быть записано вот так:

var shoppingList = [ "Eggs" , "Milk" ]

Поскольку все значения в литерале массива имеют один и тот же тип, Swift может сделать вывод, что [String] правильный тип для переменной shoppingList.

Вы можете создать массив с помощью функции Array(arrayLiteral:)

let alphabetArray = Array(arrayLiteral: "a", "b ", "c")

При необходимости создания массива, состоящего из нескольких последовательных целочисленных значений, можно использовать оператор диапазона.

*СИНТАКСИС*

Array(значение\_1...значение\_N)

Array(значение\_1..<значение\_N)

let lineArray = Array(0…9)

**Доступ и изменение массива**

Вы можете получать значения и изменять массив через его методы и свойства, или с помощью синтаксиса index.

Чтобы получить количество элементов в массиве воспользуемся read-only свойством count:

println ( "Список покупок содержит \( shoppingList . count ) элемента." )

// Выведет "Список покупок содержит 2 элемента."

Используйте свойство isEmpty типа Bool для быстрой проверки равенства свойства count нулю (0):

if shoppingList . isEmpty {

println ( "Список покупок пустой." )

} else {

println ( "Список покупок не пустой." )

}

// выведет "Список покупок не пустой."

Вы можете добавить новый элемент в конец массива путем вызова метода append(\_:):

shoppingList . append ( "Flour" )

// shoppingList теперь сдержит 3 элемента

Как альтернатива методу append, вы можете использовать оператор (+=) для добавления нового элемента в конец массива:

shoppingList += [ "Cake" ]

// shoppingList теперь содержит 4 элемента

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

// shoppingList теперь содержит 7 элементов

Значения элементов подлежащих слиянию массивов должны иметь один и тот же тип данных.

Чтобы получить значение из массива, используйте синтаксис index, передавая значения индекса в квадратных скобках сразу после имени массива:

var firstItem = shoppingList [ 0 ]

// firstItem имеет значение "Eggs"

Массивы в Swift всегда нумеруются с нуля (0).

Вы можете использовать синтаксис index для изменения существующего значения в данном индексе:

shoppingList [ 0 ] = "Six eggs"

// теперь первое значение списка имеет значение "Six eggs" в место "Eggs"

Вы также можете использовать синтаксис index для изменения сразу диапазона значений, даже если набор значений имеет разную длину, чем в указанном диапазоне. Следующий пример заменяет «Chocolate», «Cheese», и «Butter» на значение «Bananas» и «Apples» :

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

// shoppingList содержит 6 элементов

*Вы не можете использовать синтаксис Index, чтобы добавить новый элемент в конец массива.*

Для того чтобы вставить элемент на определенную позицию в массиве, нужно использовать метод insert(atIndex:):

shoppingList.insert("Syrup", atIndex: 0)

// shoppingList теперь содержит 7 элементов

// " Syrup" теперь первый элемент списка

Вызов метода insert вставляет новый элемент со значением «Syrup» в самое начало массива.

Тоже самое происходит и при удалении элементов из массива с помощью метода removeAtIndex.

let syrup = shoppingList.removeAtIndex(0)

// элемент с индексом 0 был удален

// shoppingList теперь содержит 6 элементов, Syrup удален

// константа syrup теперь равна удаленной строке "Syrup"

*Если вы попробуете использовать синтаксис index, чтобы получить или назначить новое значение индексу, который находиться за пределами существующего массива, то сработает runtime ошибка.*

Все пробелы внутри массива закрываются существующими элементами, то есть происходит сдвиг элементов по индексу. Теперь новый элемент с индексом 0 будет «Six eggs»:

firstItem = shoppingList [ 0 ]

// firstItem теперь равно "Six eggs"

 Если нужно удалить последний элемент массива, то используйте метод removeLast() вместо removeAtIndex( для избежание дополнительного вызова свойства count. Как и removeAtIndex(:), метод removeLast() так же возвращает значение удаленного элемента:

let apples = shoppingList.removeLast()

// последний элемент массива был удален

// shoppingList теперь содержит 5 элементов, и нет "Apples"

// константа apples теперь равна удаленной строке "Apples"

**Перебор массива**

Вы можете перебрать весь набор значений в массиве циклом (for — in):

for item in shoppingList {

println ( item )

}

// Six eggs

// Milk

// Flour

// Cake

// Bananas

Если вам нужно значение каждого элемента массива и его индекс, то необходимо использовать глобальную функцию enumerate для итерации по массиву. Функция enumerate возвращает кортеж, состоящий из индекса и значения элемента массива. Вы можете разложить кортеж во временные константы или переменные, как часть итерации:

for ( index , value ) in enumerate ( shoppingList ) {

println ( "Item \( index + 1 ) : \( value ) " )

}

// Item 1: Six eggs

// Item 2: Milk

// Item 3: Flour

// Item 4: Cake

// Item 5: Bananas

**Сравнение массивов**

Массивы, так же как и значения фундаментальных типов данных, можно сравнивать друг с другом . Два массива являются эквивалентными:

1) если количество элементов в сравниваемых массивах одинаково;

2) каждая соответствующая пара элементов эквивалентна .

let a1 = 1

let a2 = 2

let a3 = 3

if [1, 2, 3] == [a1, a2, a3] {

print("Массивы эквивалентны")

} else {

print("Массивы не эквивалентны")

}

Несмотря на то что в массиве [a1, a2, a3] указаны не значения, а константы, содержащие эти значения, условия эквивалентности массивов все равно выполняются .

**Многомерные массивы**

Элементами массива могут быть не только значения фундаментальных типов, но и другие массивы. Массивы, содержащие в себе другие массивы, называются многомерными. Необходимо обеспечить единство типа всех вложенных массивов.

var arrayOfArrays = [[1,2,3], [4,5,6], [7,8,9]]

Типом основного массива arrayOfArrays является [[Int]] (с удвоенными квадратными скобками с каждой стороны) .

Для доступа к элементу многомерного массива необходимо указывать несколько индексов

arrayOfArrays[2][1]

**Базовые свойства и методы массивов**

Свойство count возвращает количество элементов в массиве

var someArray = [1, 2, 3, 4, 5]

// количество элементов в массиве

someArray.count //5

Если значение свойства count равно нулю, то и свойство isEmpty возвращает true

var someArray: [Int] = [] //[]

someArray.count//0

someArray.isEmpty//true

Метод suffix(\_:) В качестве входного параметра ему передается количество элементов, которые необходимо получить . Элементы отсчитываются начиная с последнего элемента массива

var someArray = [1, 2, 3, 4, 5]

// получаем три последних элемента массива

let subArray = someArray.suffix(3) //[3,4,5]

Свойства first и last возвращают первый и последний элементы массива

someArray.first //1

someArray.last //5

С помощью метода append(\_:) можно добавить новый элемент в конец массива

Если массив хранится в переменной (то есть является изменяемым), то метод insert(\_:atIndex:) вставляет в массив новый одиночный элемент с указанным индексом

var someArray = [1, 2, 3, 4, 5]

// вставляем новый элемент в середину массива

someArray.insert(100, at: 2)

someArray

Так же как в случае изменения массива, методы remove(at:), removeFirst() и removeLast() позволяют удалять требуемые элементы . При этом они возвращают значение удаляемого элемента. После удаления индексы оставшихся элементов массива перестраиваются.

Если массив является неизменяемым (хранится в константе), то можно использовать методы dropFirst(\_:) и dropLast(\_:), возвращающие новый массив, в котором отсутствует несколько первых или последних элементов .

let someArray = [1, 2, 3, 4, 5]

// удаляем последний элемент

someArray.dropLast()

// удаляем три первых элемента

var newArray = someArray.dropFirst(3)

newArray //4, 5

При использовании данных методов основной массив someArray, с которым выполняются операции, не меняется (у нас let, а не var) . Они лишь возвращают получившееся значение, которое при необходимости может быть записано в новый параметр.

Метод contains(\_:) определяет факт наличия некоторого элемента в массиве и возвращает Bool в зависимости от результата

let intArray = [1, 2, 3, 4, 5, 6]

let resultTrue = intArray.contains(4)

Метод index(of:) сообщает индекс первого вхождения искомого элемента в рассматриваемом массиве. Так как искомый элемент может отсутствовать, метод возвращает опциональное значение. Если элемент отсутствует, то возвращается nil.

let intArray = [1, 2, 3, 4, 5, 6, 4, 5, 6]

// поиск первого вхождения элемента

let result = intArray.index(of:4) //3

Для поиска минимального или максимального элемента в массиве применяются методы min () и max (). *Сопоставлять* можно значения тех типов данных, которые являются хешируемыми, то есть в них должен присутствовать функционал вычисления хеша для значения. Большинство фундаментальных типов данных поддерживают хеширование.

Для доступа к хешу параметра достаточно вызвать указанное свойство:

var a: Float = 3.5

a.hashValue // 1 080 033 280

Чтобы изменить порядок следования всех элементов массива на противоположный, используйте метод reverse().

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Задание 4.1**

1. *Определите массив целых. Проинициализируйте его последовательными числами от 5 до 125. Удалите 5 первых элементов. Замените 5 последних элементов на два 0-ых. Выведите значения массива.*
2. *Выведите массив с использованием enumerate (кортеж – индекс и значение).*

**Наборы (множества)**

В отличие от массивов, у элементов набора нет какого-либо определенного порядка следования, важен лишь факт наличия некоторого значения в наборе. Определенное значение элемента может существовать в нем лишь единожды.

Набор создается с помощью литерала набора. В плане синтаксиса он идентичен литералу массива. Тип данных элементов набора должен быть хешируемым, то есть поддерживать протокол Hashable. При создании набора необходимо явно указать, что создается именно набор. В связи с этим необходимо:

* либо явно указать тип данных набора с использованием конструкции Set<ТипДанных>, где ТипДанных — тип элементов создаваемого набора;
* либо использовать функцию Set<ТипДанных>, которой в качестве входного передается параметр arrayLiteral, содержащий перечень элементов набора.

Если нет необходимости явно указывать тип данных значений создаваемого набора, можно использовать ключевое слово Set без конструкции <ТипДанных>.

/\* набор, созданный путем

явного указания типа \*/

var dishes: Set<String> = ["ice", "fruit"]

var dishesTwo: Set = ["ice", "fruit"]

var members = Set<String>(arrayLiteral: "Nikita", "Olga")

var membersTwo = Set(arrayLiteral: "Nikita", "Olga")

Пустой набор создается с помощью пустого литерала набора [] .

var dishes: Set<String> = ["bread", "milk"]

// создание пустого набора

var members = Set<String>()

// удаление всех элементов набора

dishes = []

**Доступ к набору и модификация набора**

Так как набор — это неупорядоченная коллекция элементов, не имеющая каких-либо индексов или ключей, обеспечивающих доступ к значениям, то использование синтаксиса сабскриптов невозможно.

Для создания нового значения в наборе применяется метод insert(\_:)

var musicStyleSet: Set<String> = []

musicStyleSet.insert("Jazz")

musicStyleSet //Jazz

В результате выполнения метода insert(\_:) возвращается массив, первый элемент которого содержит значение типа Bool, характеризующее успешность проведенной операции, и значение добавляемого элемента . В качестве первого элемента массива может быть возвращено false в том случае, когда производилась попытка добавить уже существующий в наборе элемент

Для удаления элемента из набора используется метод remove(\_:), который удаляет элемент с указанным значением и возвращает удаленное значение или nil, если удаляемого элемента не существует . Также вы можете задействовать метод removeAll() для удаления всех элементов набора

var musicStyleSet: Set<String> = ["Jazz", "Hip-Hop", "Rock"]

musicStyleSet.remove("Hip-Hop")

// удаляем несуществующий элемент

musicStyleSet.remove("Classic") //nil

musicStyleSet.removeAll() ///Set([])

Проверка факта наличия значения в наборе осуществляется методом contains(\_:) . Данный метод возвращает значение типа Bool в зависимости от результата проверки

**Базовые свойства и методы наборов**

Для двух любых наборов можно произвести следующие операции :

* получить все общие для обоих наборов элементы (intersection) - создается новый набор, содержащий значения, общие для двух наборов;

var oddDigits:Set<Int> = [2,4,6]

var differentDigits:Set<Int> = [1,2,3,4,6]

var inter = differentDigits.intersection(oddDigits).sorted()

* получить все непересекающиеся (не общие) для обоих наборов элементы (symmetricDifference);

var exclusive = differentDigits.symmetricDifference(oddDigits).sorted()

* получить все элементы обоих наборов (union);

var union = differentDigits.union(oddDigits).sorted()

* получить разницу элементов, то есть элементы, которые входят в первый набор, но не входят во второй (subtracting) .

var subtract = differentDigits.subtracting(oddDigits)

Два набора считаются эквивалентными, если у них один и тот же набор элементов. Эквивалентность наборов проверяется с помощью оператора эквивалентности (==)

var bSet: Set = [1, 3]

var copyOfBSet = bSet

if bSet == copyOfBSet {

print("Наборы эквивалентны")

}

Метод isSubset(of:) определяет, является ли один набор субнабором другого.

var aSet: Set = [1, 2, 3, 4, 5]

var bSet: Set = [1, 3]

if bSet.isSubset(of: aSet) {

print("bSet — это субнабор для aSet")

}

Метод isSuperset(of:) вычисляет, является ли набор супернабором для другого набора.

Метод isDisjoint(with:) определяет, существуют ли в двух наборах общие элементы, и в случае их отсутствия возвращает true

Методы isStrictSubset(of:) и isStrictSuperset(of:) определяют, является набор субнабором или супернабором, не равным указанному множеству

var aSet: Set = [1, 2, 3, 4, 5]

var bSet: Set = [1, 3]

if bSet.isStrictSubset(of: aSet) {

print("bSet — субнабор для aSet")

}

if aSet.isStrictSuperset(of: bSet) {

print("aSet — супернабор для bSet")

}

*Самое важное, чтобы при создании набора вы не забывали использовать ключевое слово Set, иначе в результате будет создан массив.*

**Словари**

Словарь — это неупорядоченная коллекция элементов одного и того же типа, для доступа к значениям которых используются ключи. Каждый элемент словаря состоит из уникального ключа, который указывает на данный элемент, и значения. В качестве ключа выступает не автоматически устанавливаемый индекс (как в массивах), а уникальный для словаря литерал произвольного типа, устанавливаемый разработчиком. Чаще всего ключи — это строковые литералы.

Значение словаря устанавливается с помощью литерала словаря.

var dictionary = ["one":"один", "two": "два", "three":"три"]

При попытке создания словаря с двумя одинаковыми ключами Xcode сообщит об ошибке.

Другим способом является инициализация словаря на основе двух произвольных коллекций

// базовая коллекция кортежей (пар значений)

let baseCollection = [(2, 5), (3, 6), (1, 4)]

// создание словаря на основе базовой коллекции

let newDictionary = Dictionary(uniqueKeysWithValues:baseCollection)

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

Функция zip(\_:\_:) - возвращает последовательность пар, основанную на двух базовых последовательностях. После этого сформированная последовательность передается свойству uniqueKeysWithValues инициализатора типа Dictionary. В качестве ключей результирующий словарь будет содержать значения первой базовой коллекции, а в качестве значения — элементы второй базовой коллекции.

let nearestStarNames = ["Proxima Centauri", "Alpha Centauri A", "Alpha Centauri B", "Barnard's Star", "Wolf 359"]

let nearestStarDistances = [4.24, 4.37, 4.37, 5.96, 7.78]

let starDistanceDict = Dictionary(uniqueKeysWithValues: zip(nearestStarNames, nearestStarDistances))

// ["Wolf 359": 7.78, "Alpha Centauri B": 4.37, "Proxima Centauri":

//4.24;, "Alpha Centauri A": 4.37, "Barnard's Star": 5.96]

**Взаимодействие с элементами словаря**

Ключи предназначены не только для получения значений элементов словаря, но и для их изменения

var countryDict = ["RUS":"Россия", "BEL": "Беларусь", "UKR":"Украина"]

countryDict["RUS"] = "Russian"

Для обновления значения элемента словаря можно также использовать метод updateValue(\_:forKey:) . Данный метод возвращает опциональное старое значение (nil, если значения по изменяемому ключу не существует).

Для удаления некоторого элемента (пары «ключ-значение») достаточно присвоить удаляемому элементу nil или использовать метод removeValue(forKey:), указав ключ элемента - возвращается значение удаляемого элемента. Если вы попытаетесь получить доступ к несуществующему элементу словаря, это не приведет к ошибке — Swift просто вернет nil. Это говорит о том, что любое возвращаемое значение элемента словаря — опционал.

countryDict["UKR"] = nil

countryDict.removeValue(forKey: "BEL")

let myCountry: String? = countryDict["BEL"] //nil

Для преобразования возвращаемого значения элемента из типа String? в тип String выполняется принудительное извлечение значения.

Тип данных элементов словаря содержит в себе два типа данных: тип ключа и тип значения .

*СИНТАКСИС*

var имя\_словаря: Dictionary<ТипКлюча:ТипЗначения>

var имя\_словаря: [ТипКлюча:ТипЗначения]

Для того чтобы создать пустой словарь, он должен быть инициализирован значением без элементов . Это делается с помощью конструкции [:], которая как раз и является литералом словаря, не имеющего элементов

var emptyDictionary: [String:Int] = [:]

var AnotherEmptyDictionary: Dictionary<String,Int> = [:]

С помощью конструкции [:] можно также уничтожить все элементы словаря, если присвоить ее словарю в качестве значения

**Базовые свойства и методы словарей**

Свойство count возвращает количество элементов в словаре

Если свойство count равно нулю, то свойство isEmpty возвращает true

var someDictionary: [String:Int] = [:]

someDictionary.count

someDictionary.isEmpty

При необходимости вы можете получить все ключи или все значения словаря с помощью свойств keys и values

// все ключи словаря countryDict

var keys = [String](countryDict.keys) //["BEL", "UKR", "RUS"]

// все значения словаря countryDict

var values = [String](countryDict.values) //["Беларусь", "Украина", "Россия"]

При вызове свойства keys или values Swift возвращает не массив, набор или словарь, а значение некоего типа LazyMapCollection, которое представляет собой ленивую коллекцию.

С помощью цикла for-in удобно перебирать значения коллекций . Возможна ситуация, когда требуется получить не пару «ключ-значение» из словаря, а только ключ или только значение . Для этого в кортеже на месте того элемента, который загружать не следует, необходимо вставить символ нижнего подчеркивания

var countriesAndBlocks = ["Россия": "ЕАЭС", "США": "НАТО", "Франция":"ЕС"]

for (countryName, \_) in countriesAndBlocks {

print("страна — \(countryName)")

}

Можно сделать обход только по ключам словаря:

for country in countryDict.keys {

print("Counrty code: \(country)")

}

//Counrty code: BEL

//Counrty code: UKR

//Counrty code: RUS

let somePoint = (1, 1)

switch somePoint {

case (0, 0):

print("\(somePoint) is at the origin")

case (\_, 0):

print("\(somePoint) is on the x-axis")

case (0, \_):

print("\(somePoint) is on the y-axis")

case (-2...2, -2...2):

print("\(somePoint) is inside the box")

default:

print("\(somePoint) is outside of the box")

}

// Prints "(1, 1) is inside the box»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Задание 4.2**

1. *Создайте словарь типа [String: (alpha:Character,num:Int)?], который описывает шахматную фигуру на игровом поле. В ключе словаря должно храниться имя фигуры, например «Белый конь», а в значении — кортеж, указывающий на координаты фигуры на игровом поле. Если вместо кортежа находится nil, это означает, что фигура убита (не имеет координат на игровом поле).*
2. *Добавьте в него три произвольные фигуры, одна из которых не должна иметь координат.*

*3) Создайте конструкцию if-else, которая проверяет, убита ли переданная ей фигура (элемент словаря), и выводит на консоль информацию либо о координатах фигуры, либо о ее отсутствии на игровом поле. Для получения координат переданной фигуры используйте опциональное связывание.*

**Вопросы**

1. Какие есть способы создания и инициализации массива?
2. Как сравнить массивы?
3. Как обратиться к диапазону массива?
4. Как создать множество?
5. Как получить доступ к элементам множества?
6. Какие есть способы определения словаря? Как изменить значения в словаре?