Урок 2

План урока

- Массивы и словари.

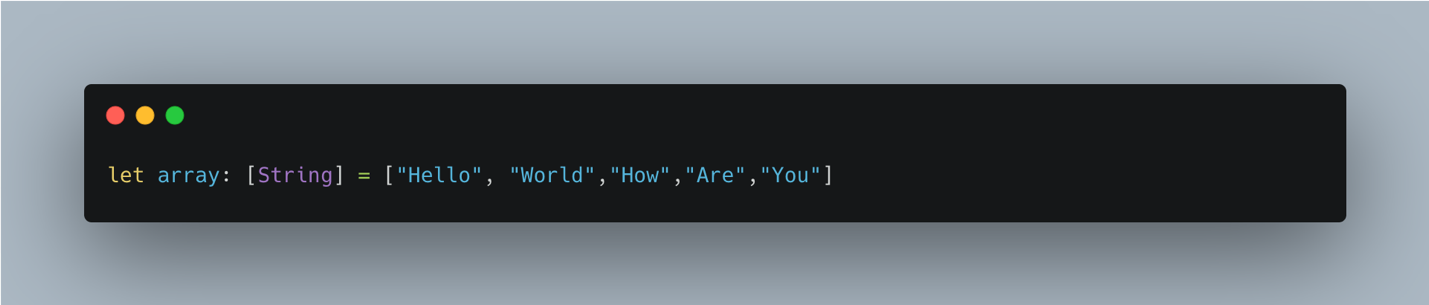
- Базовые операторы.

- Условные операторы и циклы.

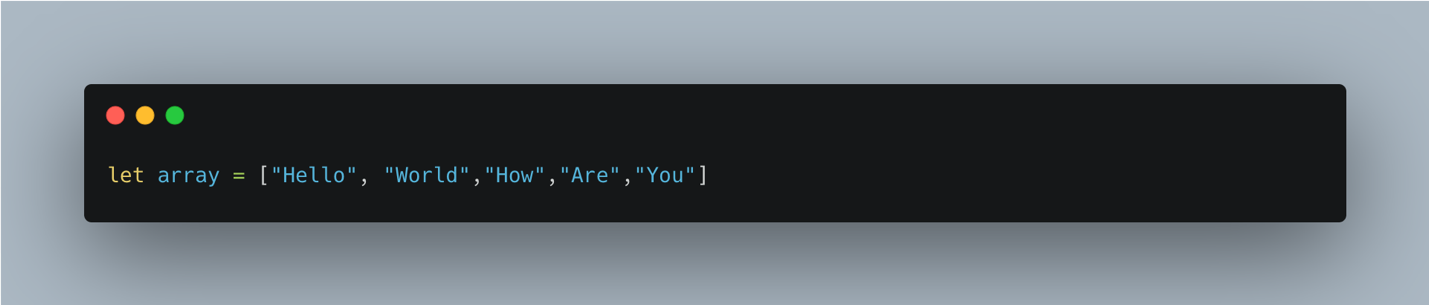
**Массивы**

Как и в любом языке массивы в swift представляют из себя упорядоченные данные. При помощи массивов мы может хранить у одной переменной разные значения.

Для того, чтобы задать массив в swift достаточно определить переменную и в скобка **[]** задать нужные значения. Так же, как и при создании переменной, мы можем указать, какой тип данных будет содержать массив.

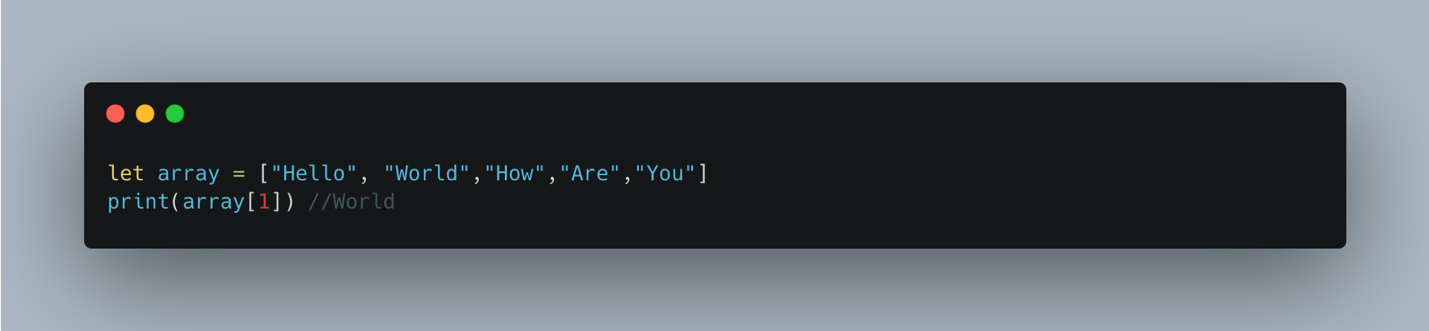


В этом примере мы создали простой массив String и задали произвольные значения. Но такую запись можно немного сократить, сделать следующим образом:



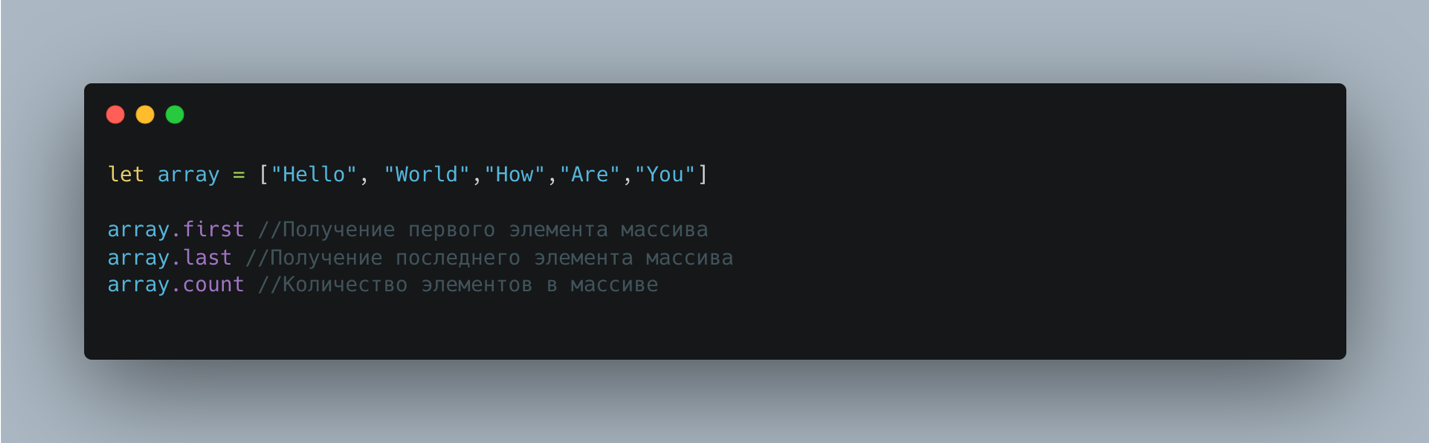
Заметьте, тут мы не указываем тип данных в массиве, но swift сам понимает, что, раз у нас все данные типа String, то соответственно и массив у нас типа [String]. Но я рекомендую все таки указывать тип массива, нередко мы будем использовать в качестве типа данных классы или структуры, и там обязательно нужно задавать тип, поэтому лучше набить руку на простых данных, чтобы потом с этим не было проблем.

Как и в любом языке, индексация элементов начинается с 0. Обратите внимание, количество элементов в массиве у нас 5, следовательно индекс последнего элемента будет 4.

Зная индекс, мы можем получить нужный нам элемент массива, к примеру:

Здесь мы пытаемся получить элемент массива под индексом 1 (второй элемент в массиве)

Так же существуют множество методов для работы с массивом, часть их них мы разберем в дальнейших практиках, в это уроке разберем основные.



В этом примере мы получаем первый и последний элемент массива, а также общее количество элементов в массиве (обратите внимание, что максимальный индекс в массиве равен количество элементов - 1)

Часто на практике бывет необходимо найти индекс по значению в массиве, для этого есть метод **firstIndex,** выглядит это следующим образом



Мы передаем в качестве аргумента в фукцию значение, индекс которого хотим узнать, но так как этого значения может и не быть внутри массива, то ответ соответственно будет опциональным, поэтому мы делаем сразу option binding.

Так же есть возможность создания опциональных массивов, для этого при создании достаточно указать **[типДанныхСожержимого]?**

В языке swift очень много методов для работы и поиску по массиву, их мы обязательно будем разбирать в дальнейших уроках.

**Set или Множество**

Не упорядоченный набор уникальных элементов. В отличие от массивов, тут уже нет индексов.

Суть типа Set заключается в том, что если мы кажем два одинаковых значения, то количество элементов будет один. К примеру, представим что у нас есть переменная items типа Set и со следующими значениям [1,2,3,4,5,5,6,3,3,5,7]. Если бы это был массив, то его длина составила бы 11 элементов, но так как это множество, и одинаковые элементы у нас не читаются больше одного раза, то длина его будет 7.

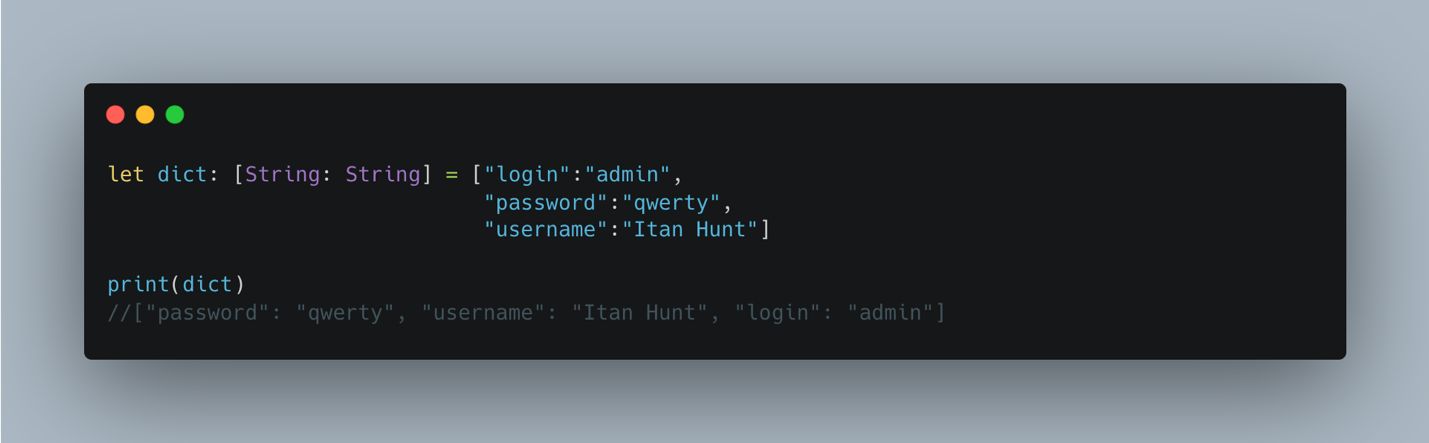
**Словари или Dictionary**

Под словарями в swift имеется ввиду структура следующего типа:

**Ключ – Значение, Ключ – Значение**

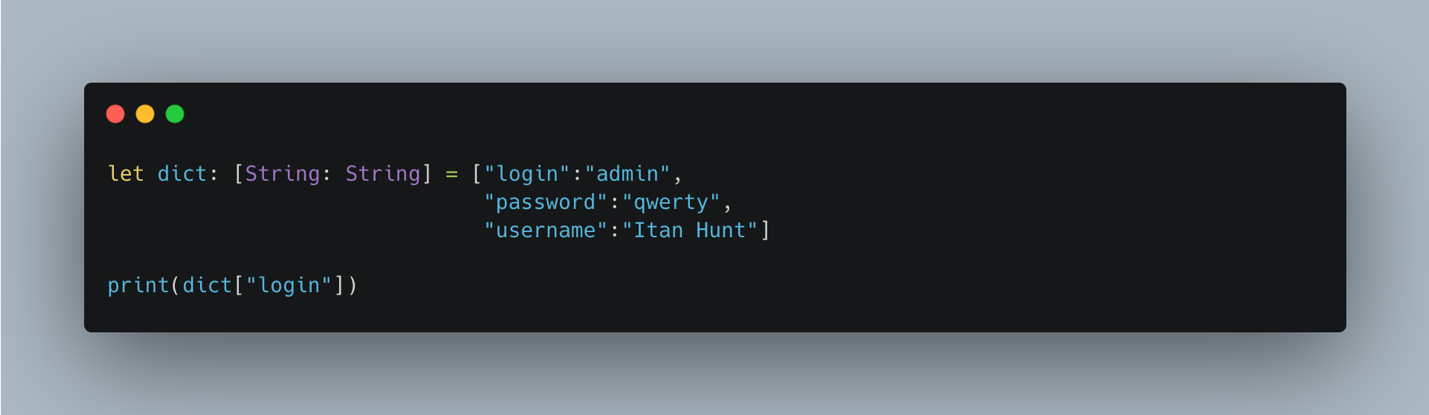
При создании мы явно указываем, какого типа будет ключ и какого типа будет значение, соответственно содержимое словаря должно соответствовать указанным типам.

Давайте рассмотрим на примере:



Обратите внимание - словари, в отличие от массивов, не упорядочены, это значит, что если мы задаем массив в порядке [1,2,4,7,3,7,2] – то в таком порядке он и выводится, но если выводим словарь, то он выводится совершенно по разному, в данном примере вывелся как (["password": "qwerty", "username": "Itan Hunt", "login": "admin"]), но это не значит, что в следующий раз выведется именно в таком порядке.

Чтобы вывести нужное значение со словаря, достаточно только указать его ключ, к примеру:



Тут мы выводим значение по ключу login. Но результат будет опциональным, так как такого ключа может и не оказаться, поэтому так же рекомендуется делать option binding.

По аналогии с опциональными массивами, можно так же создавать опциональные словари.

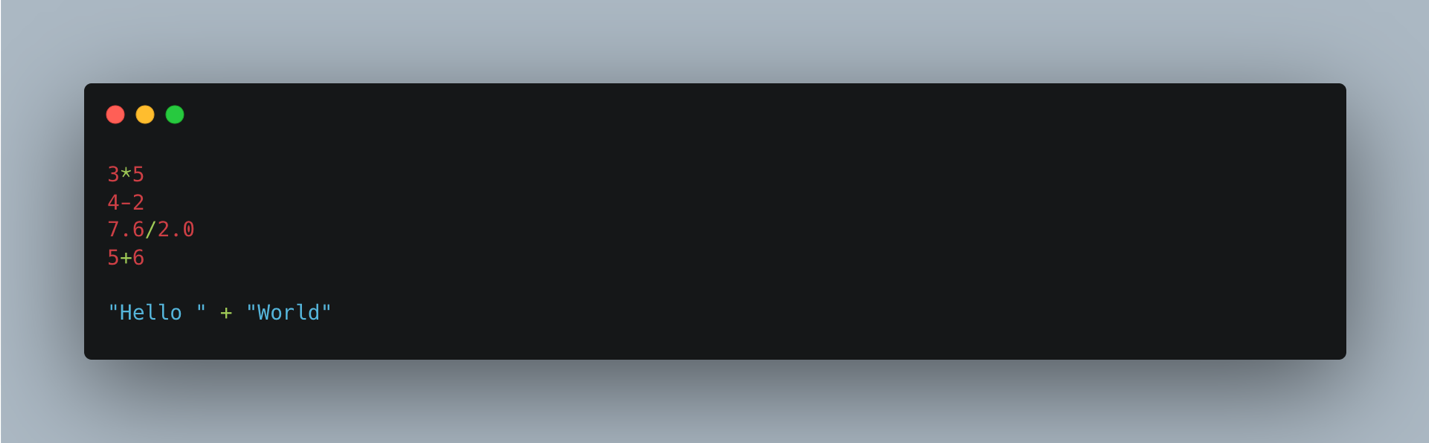
Мы так же можем делать массив словарей, для этого достаточно при создании словаря, указать тип как массив.

**Базовые операторы**

Переменные и массивы у нас есть, теперь можно совершать с ними простые операции. (+-\*/)

Совершать операции можно только с данными одного типа.

К примеру:



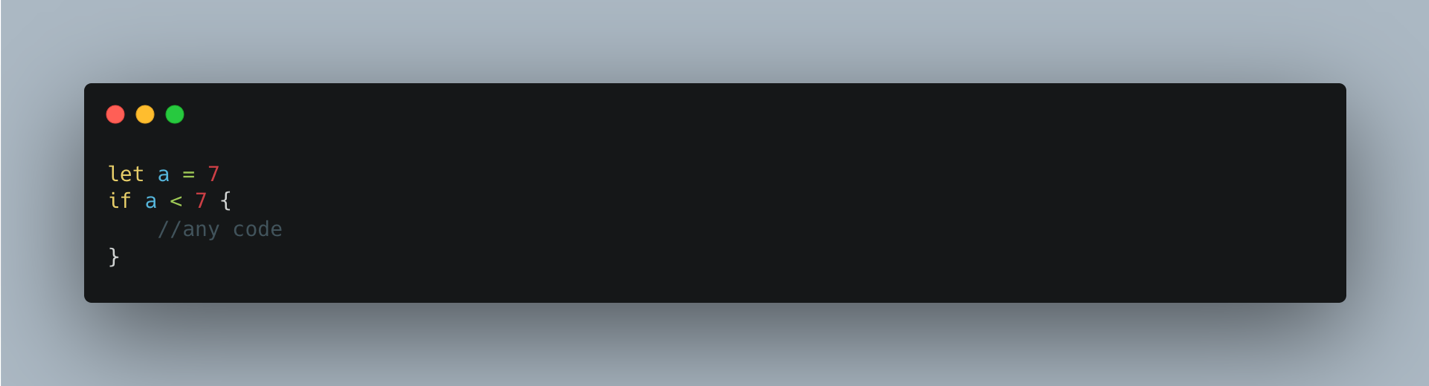
Если мы попытаемся совершить операцию с разными типами данных, то это вызовет ошибку.

В Swift как и в других языках существуют операторы сравнения, и они практически ничем не отличаются.

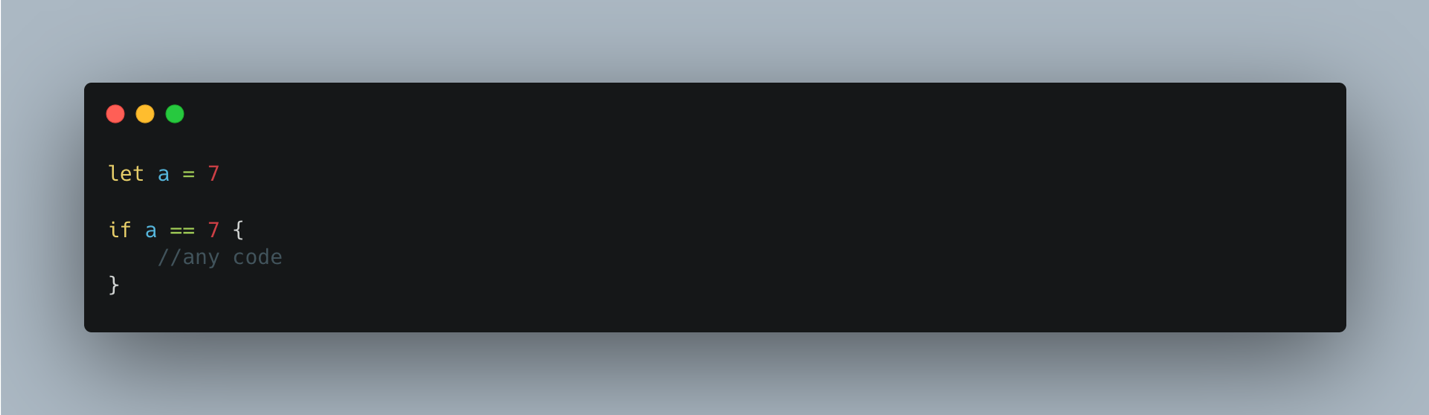
* равно (a == b)
* не равно (a != b)
* больше (a > b)
* меньше (a < b)
* больше или равно (a >= b)
* меньше или равно (a <= b)

**Условные операторы и циклы**

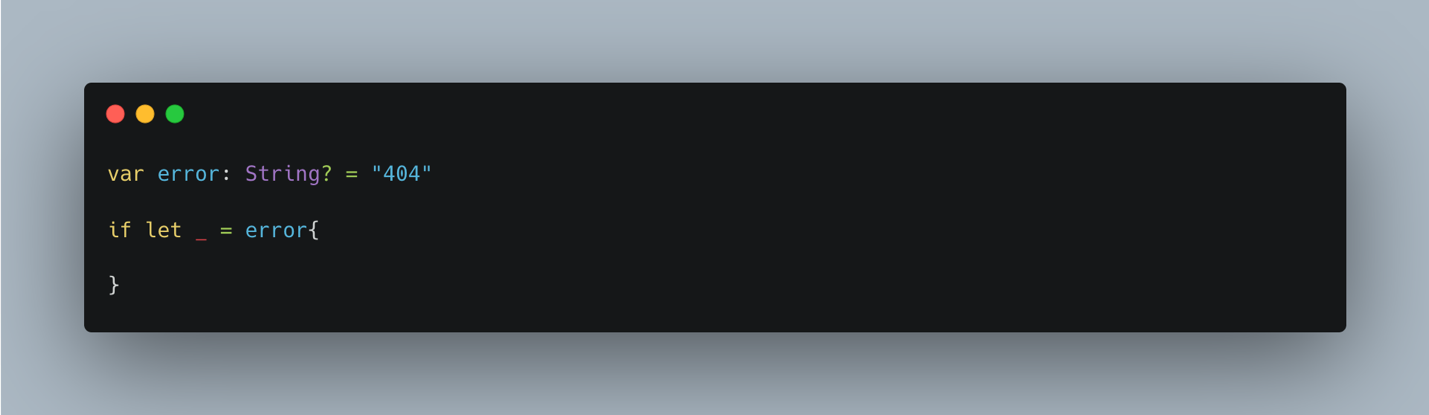
Мы уже работали с условным оператором if/else, но давайте рассмотрим его еще раз. В отличие от большинства языков, для блока условий не нужны никакие скобки. Услови мы можем писать сразу после if.



Так же мы можем проверять на равенство значений, для этого используется два знака равно (==)



Про option binding мы знаем, но представьте ситуацию, когда нам нужно сделать option binding но нет необходимости в новую переменную мы ни как не будем использовать, в таком случае мы можем поступить следующим образом:



В этом примере нам не столько нужно значение ошибки, сколько ее наличие, поэтому нет смысла создавать новую переменную, поэтому мы можем указать символ \_ в качестве пустой переменной, и, следовательно, у нас не выделится память под ненужную переменную.

Так же рассмотрим еще один оператор, по логике он похож на if/else, называется он guard. Его суть в том, что если в условии у нас пришел false дальше код не выполняется. К примеру, мы пишем запрос на сервер, и мы проверяем, пришел ли у примеру id пользователя, если он не пришел, нам нет смысла выполнять дальнейший код, и мы должны выйти их функции, или другой пример, допустим мы пытаемся найти в словаре по ключу значение и если такого нет, то нам нет смысла дальше выполнять код. Выглядит это следующим образом:

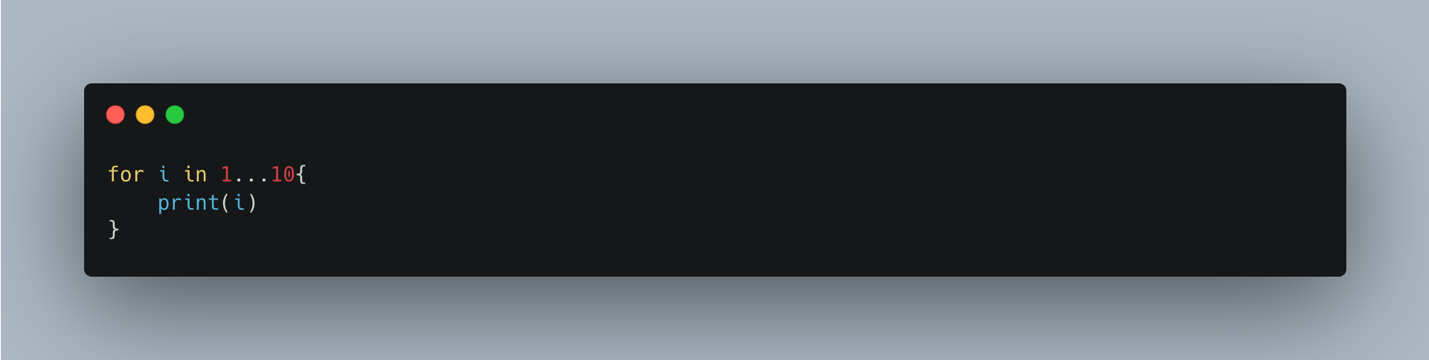


Если мы можем найти по ключу name значение в словаре, то код выполнится дальше, если нет, то попадем в return и как следствие выйдем из функции. **Важно**, используется только внутри функции, иначе xcode выдаст ошибку.

**Циклы**

Когда мы работаем с массивом данных, то так или иначе возникает необходимость в переборе всего массива. Для этого в swift как и в остальных языках есть циклы for/in, while, repeat/while

Самый распространенным является for. В нем мы можем прогнать весь наш массив и получит в каждой итерации конкретное значение и сделать те или иные действие. Синтаксис цикла слеюущий:



В данном примере мы задали диапазон от 1 до 10 включительно, в каждой итерации текущее значение будет присваиваться переменной i.

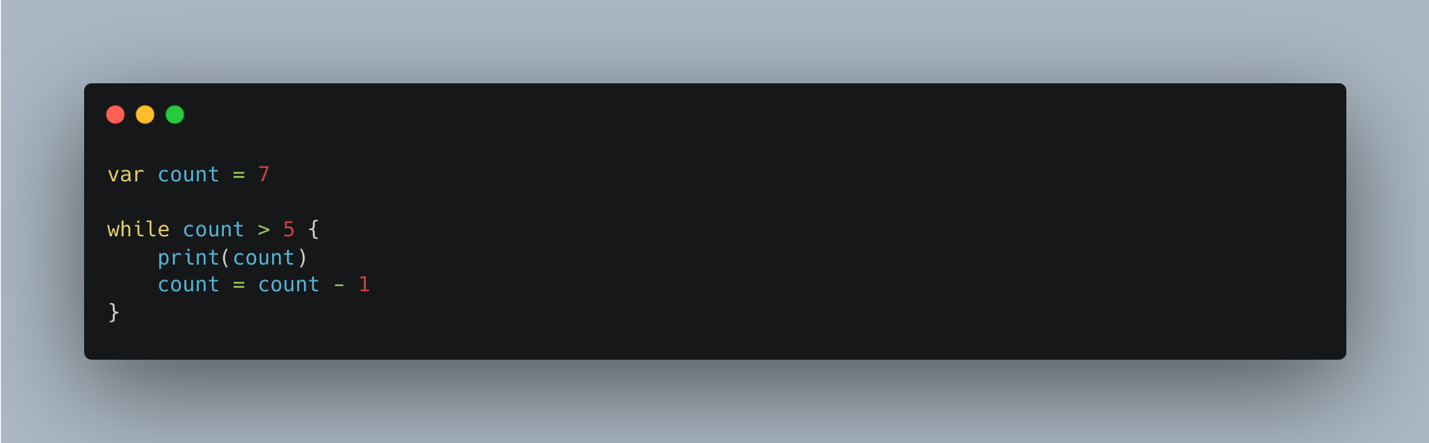
Но что если нам надо прогнать через цикл весь наш массив, по логике мы должны взять минимальный индекс 0 создать диапазон до максимального элемента -1, но swift предоставляем нам уже готовый инструмент, в качестве второго параметра мы просто можем передать сам массив, вот как это выглядит на примере:



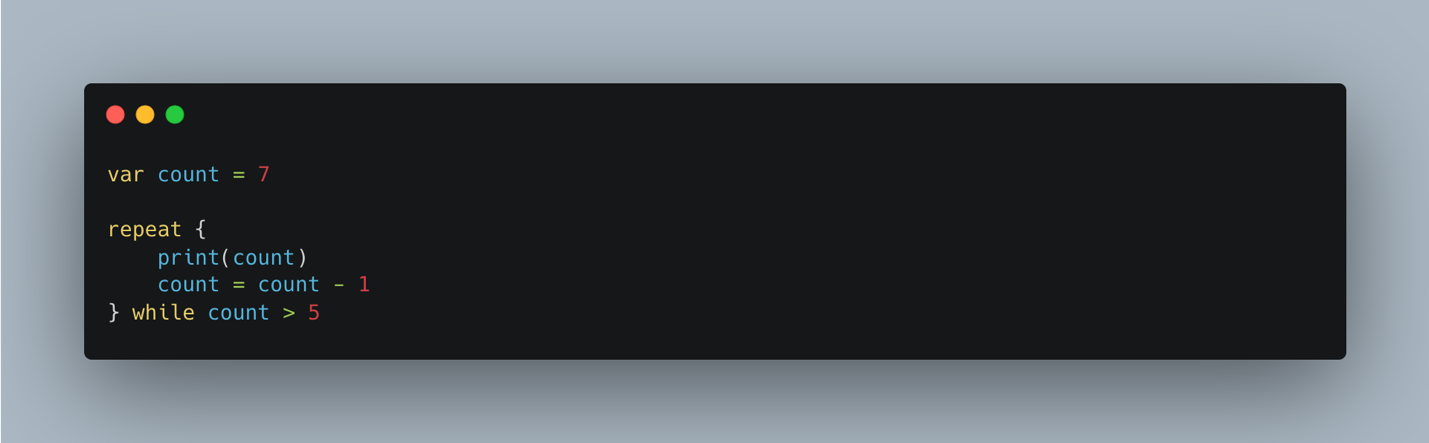
В этом примере мы проходимся по каждому пользователю, и в каждой итерации цикла пользователь у нас будет доступен в переменно user.

**Цикл while**

Цикл выполняется до тех пор, по истинно условие, поэтому с ним нужно быть очень осторожным, потому что можно вызвать цикличный код и все зависнет.



Repeat-While очень похож на обычный while только сначала мы выполняем какой-то код, проверяем условие.



ДЗ.

1 – Ознакомится с ссылками ниже.

2 – Создать массив книг (или других предметов)

3 – Найти индекс по наименованию.

4 – Удалить выбранный предмет.

5 – Добавить новое значение

6 – В цикле вывести все предметы и к каждому предмету добавить произвольную строку.

<https://docs.swift.org/swift-book/LanguageGuide/BasicOperators.html>

<https://docs.swift.org/swift-book/LanguageGuide/StringsAndCharacters.html>

<https://docs.swift.org/swift-book/LanguageGuide/ControlFlow.html>

<https://developer.apple.com/documentation/swift/set>