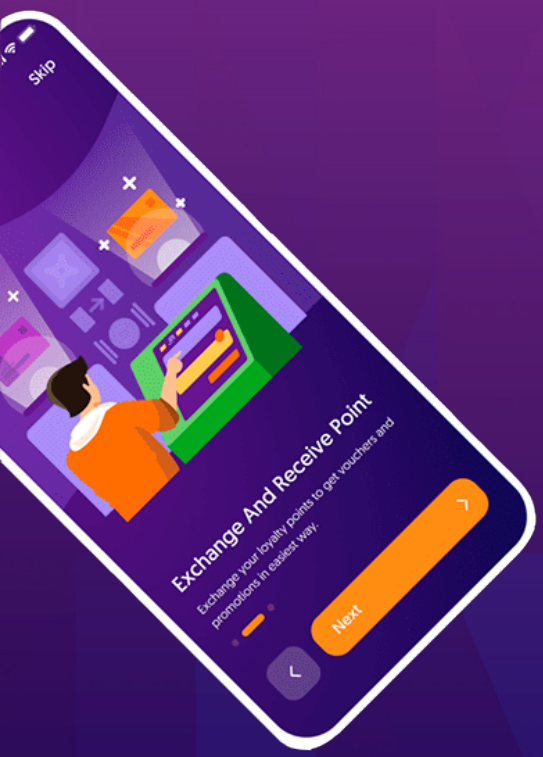
SWIFTEASY

Курс "Основы Swift"



Дмитрий Блинов

2022

Предисловие

Дорогой читатель! Меня зовут Дмитрий Блинов. Я являюсь автором курсов по программированию на современном и развивающимся языке Swift. Данная книга не является очередным учебником, хотя на данный момент книг на русском языке не так много. Этим изданием мне хотелось дополнить мои видео-уроки на YouTube, написать здесь больше теории и практических заданий. В свою очередь, видео уроки будут отвечать за освоение практической части.

Перед собой я ставил задачу создать структурированный и максимально понятный материал. Зачастую новичку достаточно сложно понять, что необходимо изучить сначала, а что потом, да и не хочется что-то упустить. Я постарался решить эту проблему. Книга разбита на курсы, которые, в свою очередь, включают в себя набор уроков. Просто начните с “Урок 0” и двигайтесь дальше! Что ж, не будем терять время! У Вас обязательно все получится!

Присоединяйтесь к нашей дружной команде:

Сайт SwiftEasy

YouTube

Группа ВКонтакте

Instagram

Краткое содержание

Курс “Основы Swift”

Урок 0
Вводный урок

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/Hx0yNxY38q8

Изучение языка программирования следует начать с создания рабочего места. Я имею ввиду выбрать модель подходящего компьютера, необходимое программное обеспечение и набор учебных материалов.

Давайте начнём с компьютера. Существует несколько вариантов:

Можно воспользоваться машиной на операционной системе Windows. Для этого необходимо создать виртуальную машину с помощью программ: VMware (vmware.com) или VirtualBox (virtualbox.org).

Рис 1.0.1 VMware и VirtualBox

Будет необходимо установить образ операционной системы семейства macOS. Но стоит отметить, что не на каждом компьютере удается её запустить, это связано с различием в используемым оборудовании (hardware). На мой взгляд этот путь не даст полноценных возможностей, ввиду того что компания Apple не приветствует это и пытается с этим бороться, вплоть до блокировки аккаунта разработчика.

Наиболее актуальный вариант - найти компьютер компании Apple. К сожалению, по российским меркам они достаточно дорогие, но можно найти вариант с рук. Нам подойдёт любой, будь то MacBook Air, MacBook Pro, iMac, Mac Pro, Mac mini.

Рис 1.0.2 MacBook Pro

Я склоняюсь больше к семейству ноутбуков MacBook Pro. Во-первых, его можно взять с собой, во-вторых, в домашних условиях его можно оснастить дополнительным монитором для удобства. В 2022-м году обязательно нужно обратить внимание на объем оперативной памяти (не менее 16 ГБ) и наличие SSD диска (это значительно увеличивает скорость работы, не менее 256 ГБ). После приобретения компьютера нам понадобится бесплатная программа Xcode, её можно найти в App Store. И вот мы готовы написать первую программу!

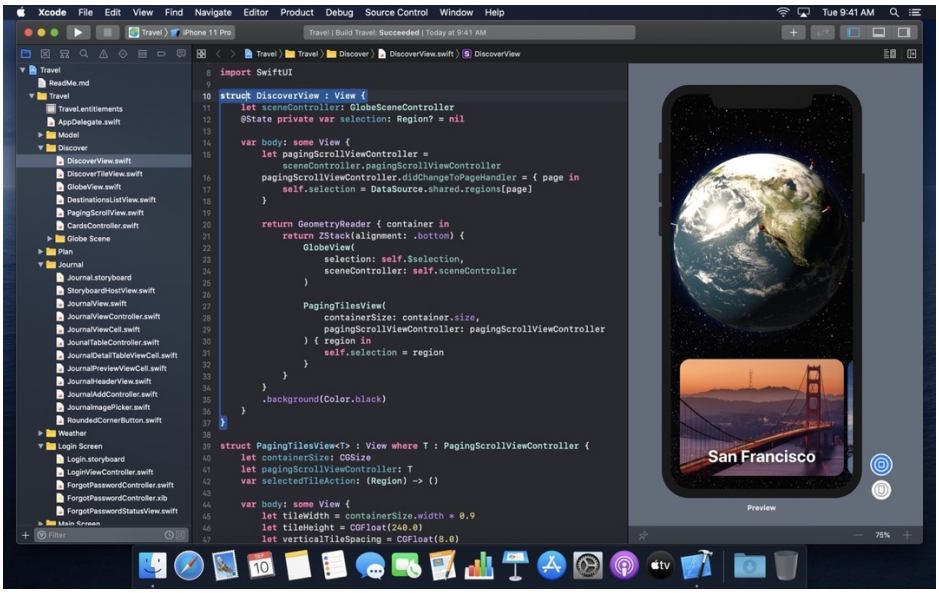


Рис 1.0.3 Окно программы Xcode

Третий вариант для тех, кто ещё не решился купить необходимое оборудование, но на начальном этапе хочет проверить свои силы в программировании. Существуют сайты online-компиляторы, на которых можно написать свой код и запустить его на выполнение. Введите в

поисковую строку сайта google.com запрос «Swift online compiler» и увидите список таких сайтов.

Что касается учебной литературы: используйте данное учебное пособие совместно с видео уроками на YouTube, на раннем этапе этого будет достаточно. Но со временем обязательно изучайте информацию и из других источников, например, изучите официальную документацию по актуальной версии языка Swift книги «The Swift Programming Language».

Рис 1.0.4 Книга «The Swift Programming Language»

Урок 1
Устройство iPhone и программа

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/W6-ug5Qy-CA

Постараемся абстрагироваться от побочных процессов, происходящих в компьютере, и сосредоточим наше внимание только на самом важном для нас.

Любая программа – это алгоритм (набор действий) над чем-либо для получения конечного результата. Давайте представим, что мы пишем калькулятор, который складывает два числа и выдает результат операции. Эти числа нашей программе необходимо где-то хранить.

Различают 2 вида памяти: оперативную и постоянную. Постоянная память используется для хранения файлов, их открытия и изменения по необходимости. Оперативная память – это память, которая используется устройством (компьютером или телефоном) для вычислений при работе программы. Такая память работает намного быстрее и имеет значительно меньший объем по сравнению с постоянной.

Для хранения чисел a, b и c программе потребуется оперативная память. Объем оперативной памяти ограничен и нам, как разработчикам, стоит позаботиться об ее рациональном использовании.

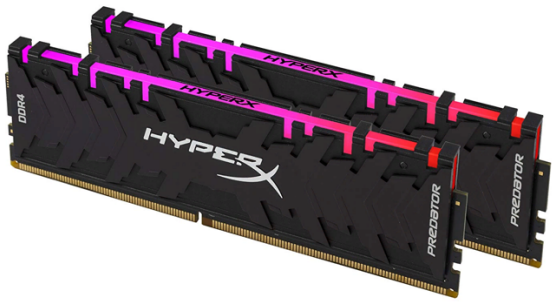


Рис 1.1.1 Две планки оперативной памяти

Сама память представляет из себя набор ячеек. Одна ячейка может принимать 2 состояния "1" или "0". Такую ячейку называют битом. Принято объединять ячейки в блоки (на рис. 1.0 такой блок назвали ячейкой) большего объема. Такой блок содержит 8 бит и называется байтом.

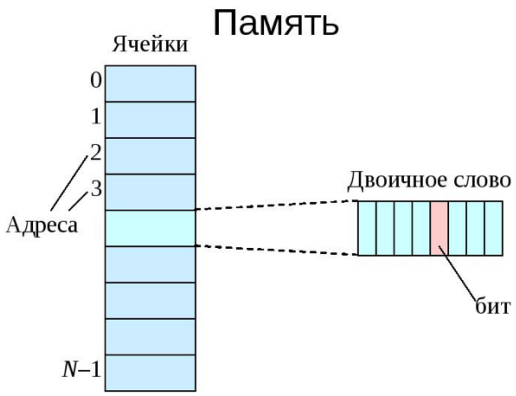


Рис 1.1.2 Организация памяти

В международном сообществе принято использовать общепринятые единицы измерения объема данных. Ниже приведена таблица обозначений.

Измерения в байтах								
ГОСТ 8.417—2002			Приставки СИ		Приставки МЭК			
Название	Обозначение	Степень	Название	Степень	Название	Обозначение		Степень
байт	Б	10^0	—	10^0	байт	B	Б	2^0
килобайт	Кбайт	10^3	кило-	10^3	кибибайт	KiB	КиБ	2^{10}
мегабайт	Мбайт	10^6	мега-	10^6	мебибайт	MiB	МиБ	2^{20}
гигабайт	Гбайт	10^9	гига-	10^9	гибибайт	GiB	ГиБ	2^{30}
терабайт	Тбайт	10^{12}	тера-	10^{12}	тебибайт	TiB	ТиБ	2^{40}
петабайт	Пбайт	10^{15}	пета-	10^{15}	пебибайт	PiB	ПиБ	2^{50}
эксабайт	Эбайт	10^{18}	экса-	10^{18}	эксбибайт	EiB	ЭиБ	2^{60}
зеттабайт	Збайт	10^{21}	зетта-	10^{21}	зебибайт	ZiB	ЗиБ	2^{70}
йоттабайт	Ибайт	10^{24}	йотта-	10^{24}	йобибайт	YiB	ЙиБ	2^{80}

Рис 1.1.3 Единицы измерения памяти

Последние модели iPhone (11 и 12) имеют по 4 ГБ оперативной памяти (LPDDR4).

Урок 2
Обзор среды Xcode. Playground.

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/SarnNbsXKq4

Для запуска необходимо найти иконку приложения Xcode и нажать на нее.

Рис 1.2.1 Иконка приложения Xcode

Перед нами появится приветственное окно. Имеется возможность создать новый проект, склонировать существующий из Git-репозитория, открыть проект или файл.

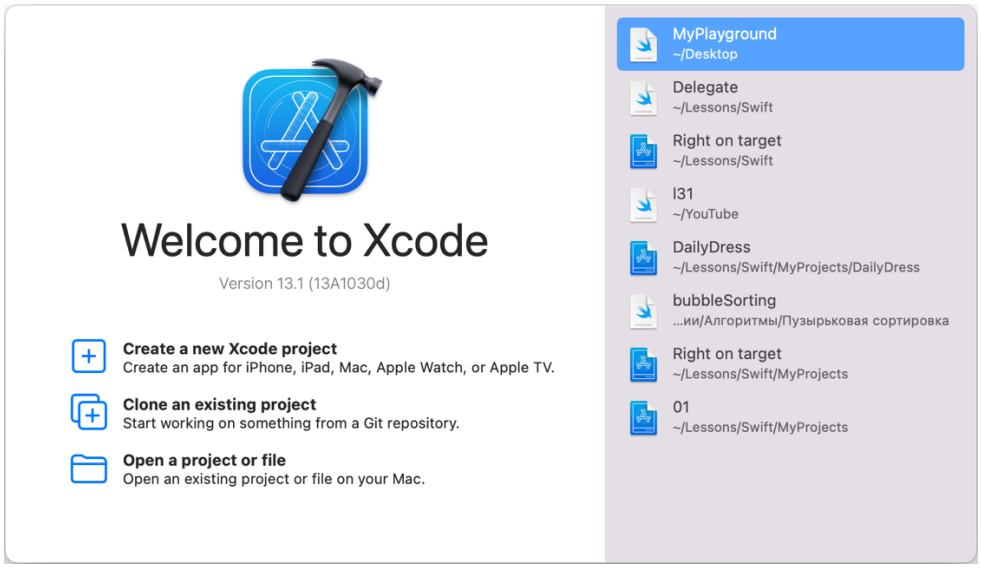


Рис 1.2.2 Приветственное окно Xcode

При выборе кнопки “Create a new Project” необходимо выбрать платформу, в нашем случае это iOS. И Приложение “App”.

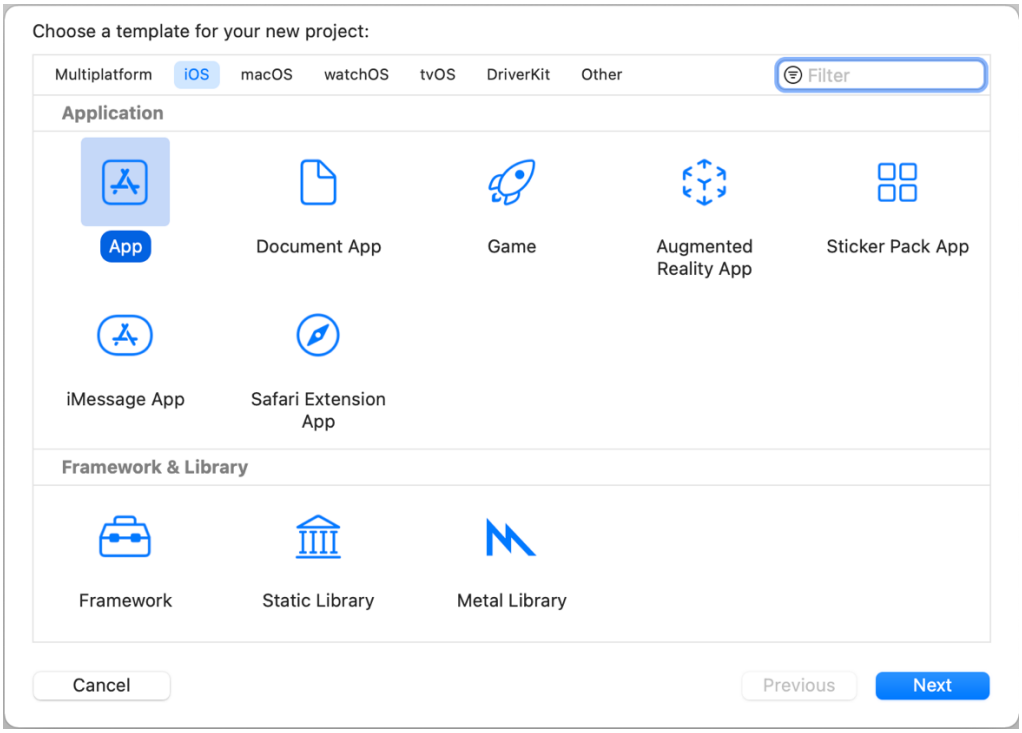


Рис 1.2.3 Выбор шаблона для нового проекта

После чего в поле “Product Name” необходимо дать имя новому проекту. Выбрать аккаунт, от которого будет разрабатываться приложение в поле “Team”. А также некоторые другие настройки, которые мы изучим более подробно немного позже. Сразу после выбора расположения директории для нового проекта откроется весь спектр возможностей Xcode для написания крутых приложений.

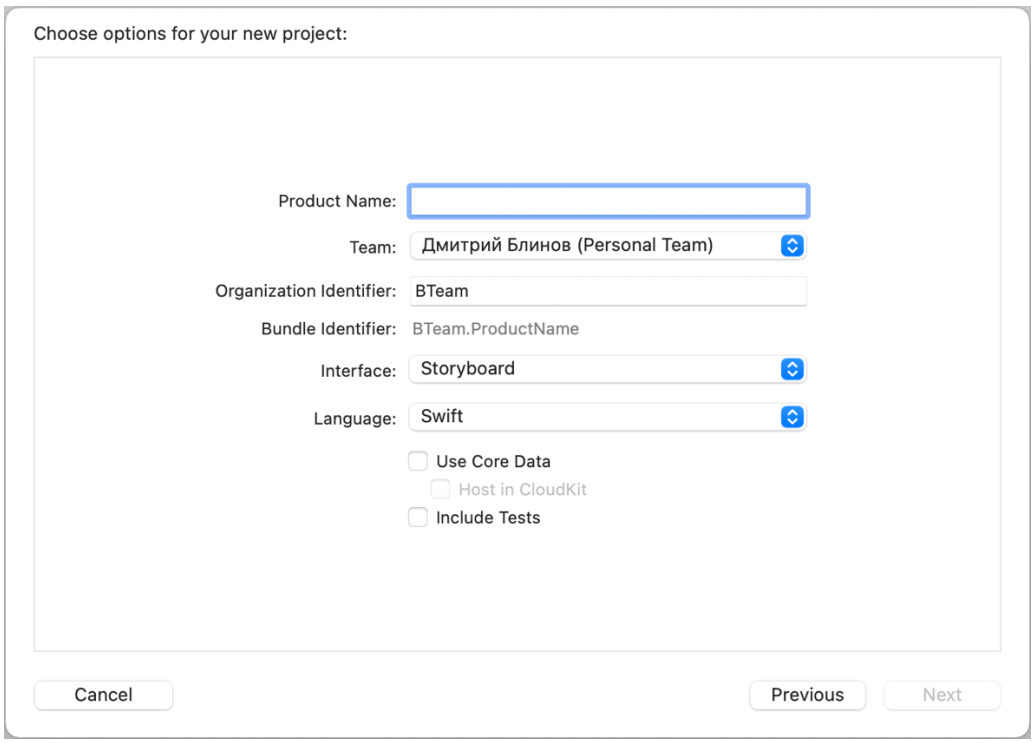


Рис 1.2.4 Параметры нового проекта

Данный режим позволяет разрабатывать полноценные приложения.

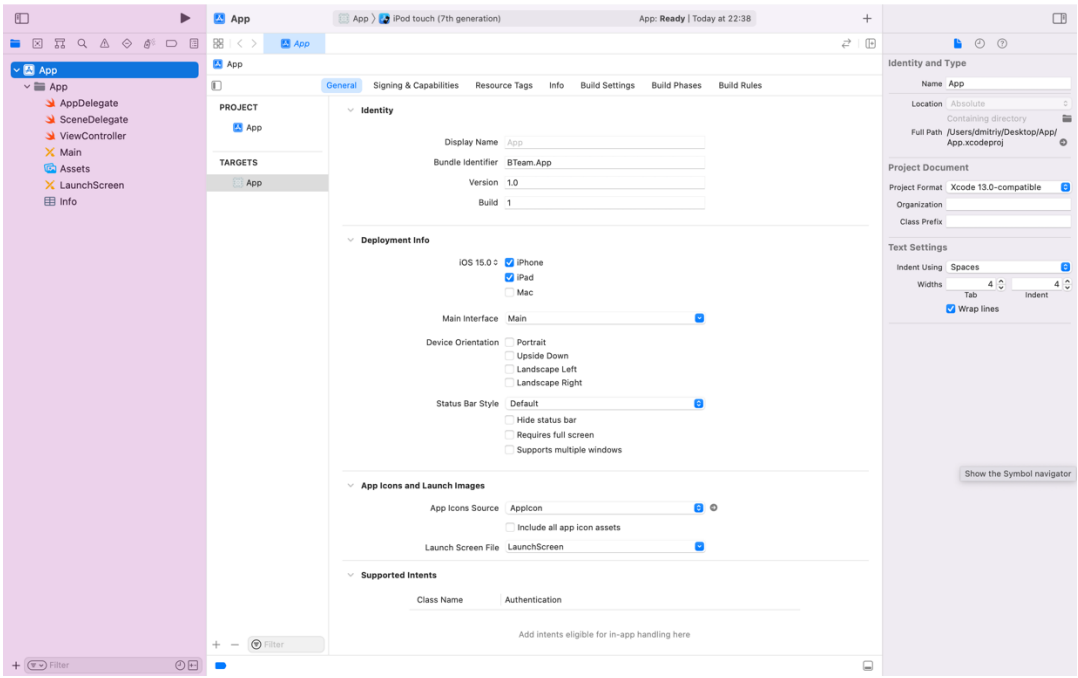


Рис 1.2.4 Окно созданного проекта

Но в данный момент нас интересует работа с Playground. Playground - это учебный удобный инструмент, который позволяет писать и тестировать участки кода, не создавая проект. Это очень удобно, особенно на ранних этапах изучения языка Swift.

Для создания файла Playground необходимо запустить Xcode и начать кнопку “File”, “New”, а затем “Playground”.

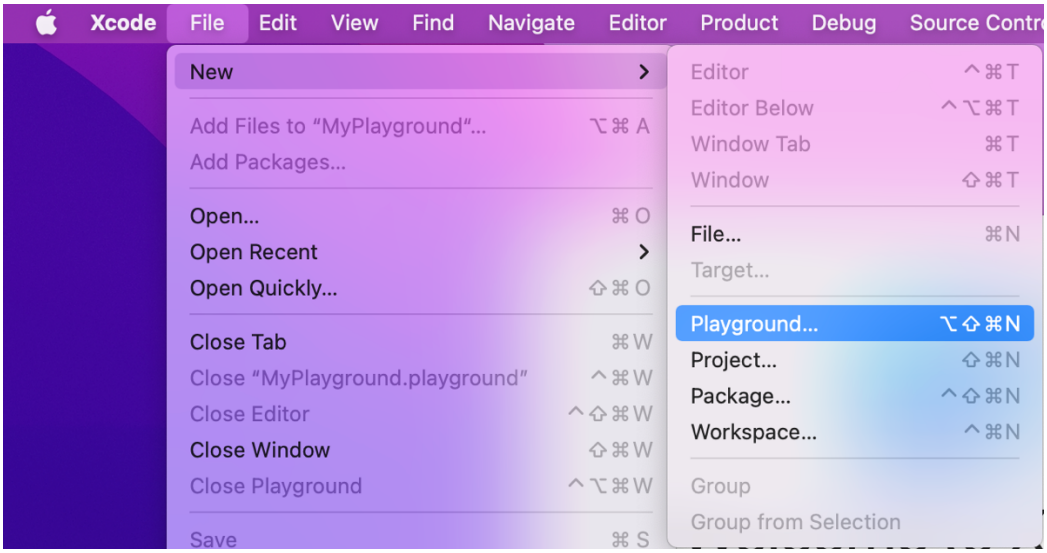


Рис 1.2.5 Окно создания Playground

В появившемся окне необходимо выбрать шаблон, в нашем случае это “Blank”.

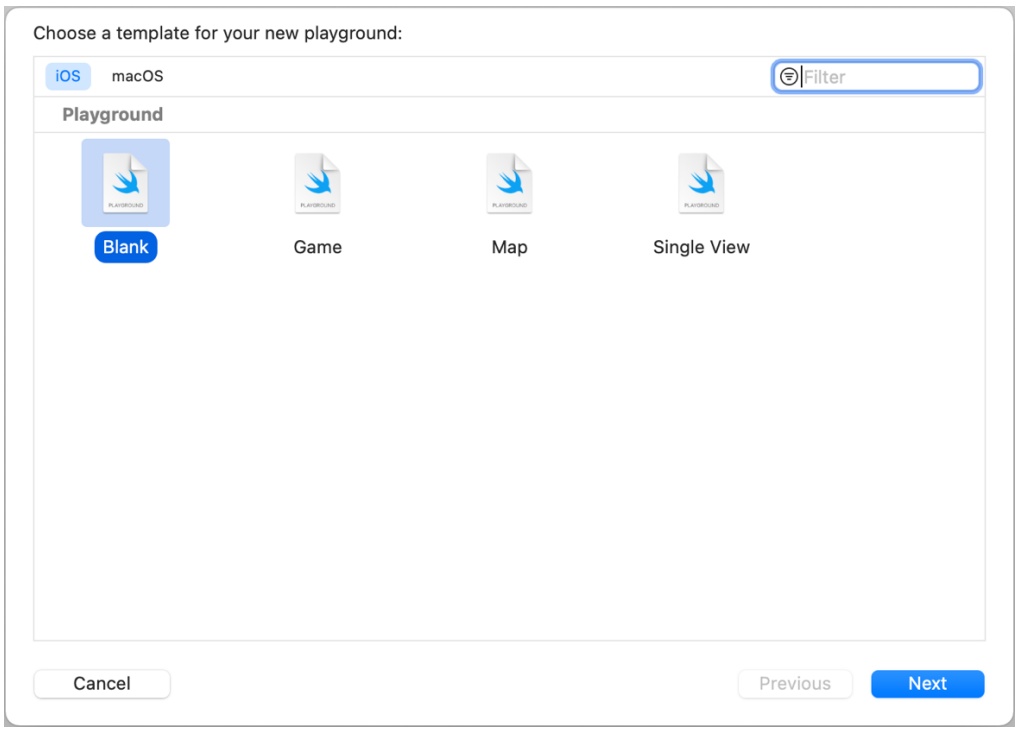


Рис 1.2.6 Выбор шаблона проекта

Затем выбрать место для хранения нашего файла Playground и нажать кнопку “Create”.

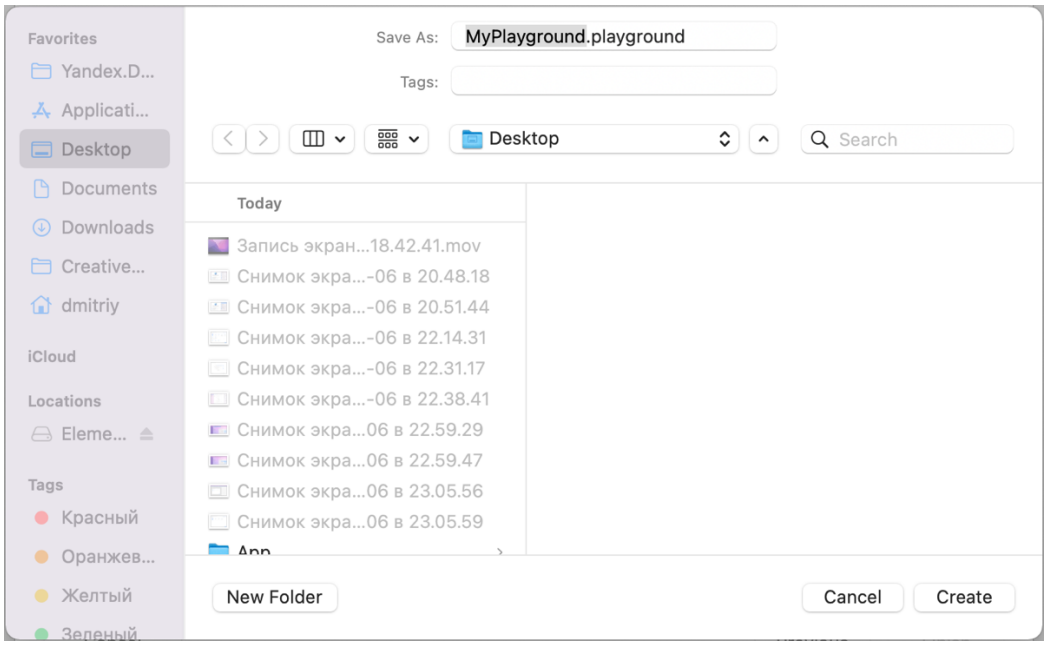


Рис 1.2.7 Сохранение файла проекта

Перед нами открылось окно, в котором мы уже сейчас попробуем написать наши первые строки кода.

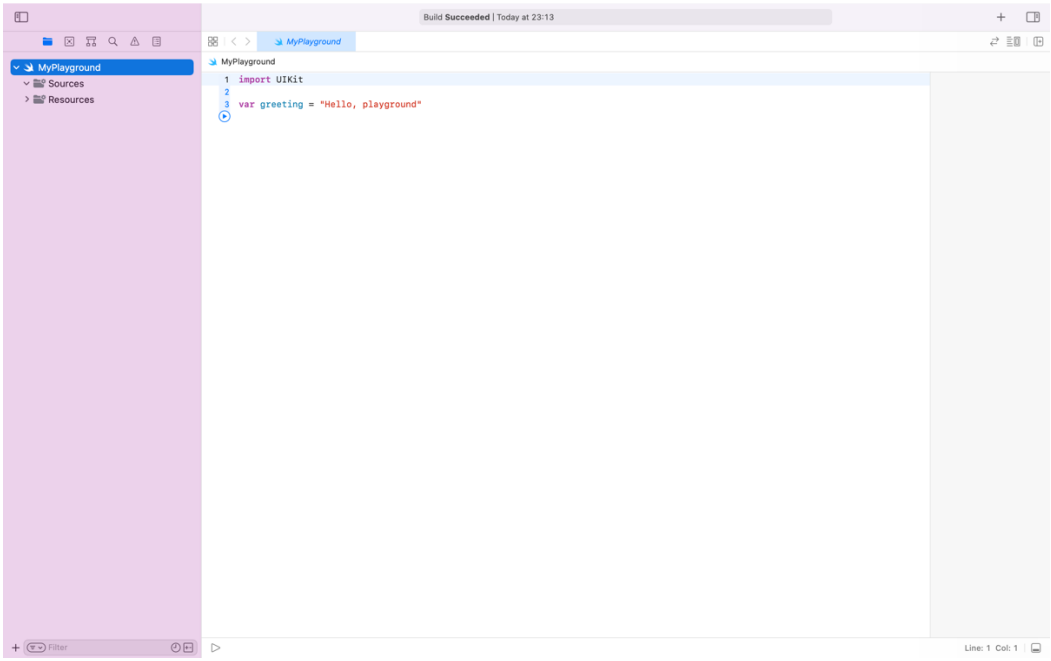


Рис 1.2.8 Окно созданного проекта

Домашнее задание!

Создайте проект Playground и дайте название файлу проекта на свое усмотрение. Затем попробуйте изменить значение переменной “greating” на число, например, “1432”. Для этого сотрите текст после символа “=”. Должно получиться так:

var greating = 1432

И запустите проект нажатием на кнопку в виде треугольника. Надеюсь, у Вас получилось =)

Обязательно поэкспериментируйте и с другими значениями.

Урок 3
Переменные и константы

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/xqysBullHcM

Для хранения данных в языке Swift существует 2 вида хранилищ: переменные и константы. Их необходимо воспринимать как некая “коробка” определённого объема памяти, в которую мы можем положить число, символ, строку и другие данные.

Переменная - это хранилище данных определённого типа, значение которого может быть изменено.

Понятие типа данных будет изучено в следующем уроке.

Константа - это хранилище данных определённого типа, которому один раз присваивается значение и больше быть изменено оно не может. Такой механизм служит для защиты от непреднамеренного изменение данных.

Каждое хранилище данных обладает тремя свойствами:

- Имеет имя
- Для хранилища определён тип данных
- Хранилище имеет значение

Рассмотрим это на примере объявления переменной.

var myNumber = 4

А теперь разберем все по полочкам, что тут происходит. Первоначально мы указываем, что мы хотим создать: переменную или константу. В данном случае мы объявили переменную конструкцией “var”. Затем следует имя переменной, оно может быть любым, лишь бы не использовало служебных слов типа того же “var”, оно уже занято.

Затем следует сивол “=”, он обозначает, что левой части (переменной с имением, которое мы ей присвоили) присваивается значение расположенное справа, то есть цифра 4.

Домашнее задание!

Создайте проект Playground. Создайте переменную и константу с произвольными именами. Присвойте каждой переменной и константе значения. Причем пусть константа будет содержать тестовое значение, а переменная число.

Урок 4
Комментарии

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/orTT6Fpi7ks

Зачастую исходный код программ достаточно сложен и большой по объему. Кроме того, над одной программой сразу может трудиться несколько человек.

Для того чтобы было легче разбираться в исходных кодах программ существуют комментарии.

Комментарий – это конструкция из нескольких символов, которая позволяет писать текст, игнорируемый компилятором.

В языке Swift различают два вида комментариев: однострочные и многострочные.

Рассмотрим синтаксис на примере:

```
//функция печати на экран
print("Hello!")
```

Однострочный комментарий состоит из символов "//", после которых написанный текст игнорируется до начала следующей строки.

Многострочный комментарий позволяет создавать многострочные подписи. Для этого используют конструкцию символов "/*" в начале комментария и "*/" в конце.

Рассмотрим синтаксис на примере:

```
/*
функция
печати
на
экран
*/
print("Hello!")
```

Урок 5
Типы данных

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/Av90PpNKwSs

Давайте познакомимся с важнейшей темой – "Типы данных". Как мы изучили ранее, к контейнерам данных относятся переменные и константы. В них мы можем положить данные. По своей природе данные различны (можно записать число, а можно строку). Отсюда происходят типы данных.

Тип данных – это множество значений, над которым можно производить определенные операции.

В языке Swift существует два варианта определения для контейнера данных типа данных:

- Явное
- Неявное

С неявным определением типа мы уже сталкивались. Давайте создадим переменную и запишем в нее текст.

var greeting = "Hello, playground"

В таком случае Xcode автоматически определяет, что присваиваемый текст *"Hello, playground"* является типом String. Поэтому при объявлении переменной *greeting*, ей присвоился тип String.

В случае с неявным определением типа используется чуть более сложная кодовая конструкция:

var greeting: String = "Hello, playground"

Как Вы уже заметили, после написания имени контейнера ставится двоеточие и указывается тип.

Для того чтобы узнать тип той или иной переменной существует функция type. Используйте следующую конструкцию:

type(of: greeting)

где *greeting* – имя переменной.

Домашнее задание!

Создайте проект Playground. Объявите любой контейнер данных с произвольным именем. Присвойте ему значения, явно определив тип данных. Передайте в функцию печати на экран print() результат работы функции type(of: от_вашей_переменной).

Урок 6
Числовые типы данных. Целочисленные типы.

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/WkhQST6qSnU

Изучение данной темы необходимо начать с изучения того, какими бывают числа, на какие группы их делят. Числа бывают целые и дробные.

Целые числа — это числа, у которых есть дробная часть. Например, 15,1000 или -21.

Для хранения таких чисел существуют определённые типы данных, например, Int. Но не стоит забывать, что при объявлении хранилищ данных выделяется определённый объем памяти. От этого объема зависит ширина диапазона значений, принимаемых контейнером данных.

Рассмотрим это на примере типа Int8. Переменная или константа данного типа используют 8 бит памяти (тех самых яичек, которые мы рассмотрели Уроке 1). Представьте, что мы поставили ячейки друг за другом.

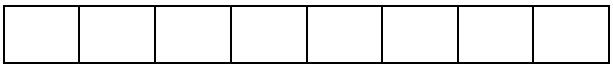


Рис 1.6.1 Восемь ячеек памяти подряд (байт)

Теперь такая конструкция бит может принимать от всех нулей в каждом бите, до всех единиц.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
•	•	•					
1	1	1	1	1	1	1	1

Рис 1.6.2 Возможные комбинации байта

Для того, чтобы подсчитать количество значений, который может принять такой блок ячеек необходимо воспользоваться формулой.

Количество состояний вычисляется по следующим образом: $C = 2^n$, где C – количество состояний, 2-разрядность системы счисления, n – количество ячеек (бит).

Затем представим, что у нас есть числа отрицательные и положительные, а ещё ноль. Половину значений займут отрицательные числа, другие положительные. Под ноль заберём место у положительных чисел.

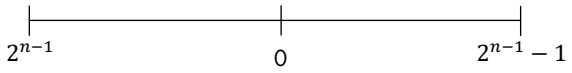


Рис 1.6.3 Возможные значения типа

Таким образом, для Int8 минимальное значение -128, максимальное 127. Мы рассмотрели знаковый тип данных.

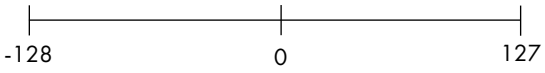


Рис 1.6.4 Возможные значения типа Int8

Знаковый тип - этот тип, в котором значение переменных и констант могут быть нулём, отрицательными и положительными числами. К таким типам относятся Int, Int8, Int16, Int32, Int64.

Беззнаковый тип данных - это тип, в котором значение переменных могут быть только положительными или нулем (UInt, UInt8, UInt16, UInt32, UInt64). Аналогично знаковому типу рассчитывается и без знаковый, только не выделяется память для отрицательных чисел. Название типа пошло от двух английских снов (Unsigned Integer) по одной букве первого слова и трём второго.

Рис 1.6.5 Возможные значения беззнакового типа

Таким образом, для Int8 минимальное значение -128, максимальное 127. Мы рассмотрели знаковый тип данных.

Рис 1.6.6 Возможные значения беззнакового типа UInt8

Тип данных	Минимальное значение	Максимальное значение
Int	−9223372036854775808	9223372036854775807
Int8	−128	127
Int16	−32768	32767
Int32	−2147483648	2147483647
Int64	−9223372036854775808	9223372036854775807
UInt	0	18446744073709551615
UInt8	0	255
UInt16	0	65535
UInt32	0	4294967295
UInt64	0	18446744073709551615

Рис 1.6.7 Целочисленные типы данных

Домашнее задание!

Создайте проект Playground. Поэкспериментируете со знаковыми и беззнаковыми типами. Изучите диапазоны возможных значений.

Урок 7
Дробные типы данных. Математические операции.

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/8vKRcA8bLeE

Существуют случаи, когда нам необходимо вычислять значения с точностью до нескольких знаков после запятой, но уже тип Int нам совершенно не подходит.

В связи с этим в языке Swift существуют два фундаментальных типа данных Float и Double.

Данные типы данных позволяют хранить числа с дробной частью. А различие их между собой лишь в том, сколько бит памяти они расходуют (сколько знаков после точки мы можем записать).

Float позволяет хранить 32-битное число с плавающей точкой, содержащее до 6 знаков в дробной части.

Double позволяет хранить 64-битное число с плавающей точкой, содержащее до 15 знаков в дробной части.

Для выполнения математических операций используют стандартный набор символов, известный нам из курса математики.

Символ	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление

Рис 1.7.1 Математические операции

Домашнее задание!

Создайте проект Playground. Создайте одну переменную типа Int, вторую типа Double. Преобразуйте переменную типа Int в Double и произведите все 4 математических операции над ними.

Урок 8
Строки и символы

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/5fBEo6O-B1M

Обмен информации не заканчивается на использовании только цифр. Зачастую или даже повсеместно мы используем текстовую информацию. Будь то отдельные символы или целые слова.

Для хранения информации весь текст делится на символы. В свою очередь, каждый символ записываются в память. Возникает вопрос: “Как же хранить символ (букву) в ячейке данных (допустим, размером в 1 Байт)?”.

Мировое сообщество придумало множество стандартов, которые описывают принципы кодирования символов.

Кодирование символов - это процесс сопоставления каждому символу из множества символов определённого числа. То есть создается список используемых символов, и каждому символу присваивается свое число (кодовая точка). Одним из таких стандартов является Unicode.

Unicode

16-битовая версия (2^16 = 65 536 значений), где кодируются все современные алфавиты.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	402 Ђ	403 Ѓ	201A ‚	453 ѓ	201E „	2026 …	2020 †	2021 ‡	20AC €	2030 ‰	409 Љ	2039 ‹	40A Њ	40C Ќ	40B Ћ	40F Џ
90	452 ђ	2018 ‘	2019 ’	201C “	201D ”	2022 •	2013 –	2014 —	□	2122 ™	459 љ	203A ›	45A њ	45C ќ	45B ћ	45F џ
A0	A0	40E Ў	45E ў	408 Ј	A4 ¤	490 Ґ	A6 ¦	A7 §	401 Ё	A9 ©	404 Є	AB «	AC ¬	AD -	AE ®	407 Ї
B0	B0 °	B1 ±	406 І	456 і	491 ґ	B5 µ	B6 ¶	B7 ·	451 ё	2116 №	454 є	BB »	458 ј	405 Ѕ	455 ѕ	457 ї
C0	410 А	411 Б	412 В	413 Г	414 Д	415 Е	416 Ж	417 З	418 И	419 Й	41A К	41B Л	41C М	41D Н	41E О	41F П
D0	420 Р	421 С	422 Т	423 У	424 Ф	425 Х	426 Ц	427 Ч	428 Ш	429 Щ	42A Ъ	42B Ы	42C Ь	42D Э	42E Ю	42F Я
E0	430 а	431 б	432 в	433 г	434 д	435 е	436 ж	437 з	438 и	439 й	43A к	43B л	43C м	43D н	43E о	43F п
F0	440 р	441 с	442 т	443 у	444 ф	445 х	446 ц	447 ч	448 ш	449 щ	44A ъ	44B ы	44C ь	44D э	44E ю	44F я

Рис 1.8.1 Часть символов Unicode-таблицы

На каждый на данный момент этот стандарт содержит около 140 000 символов.

Существует два основных типа данных:

- Character (предназначен для хранения символов)
- String (предназначен для хранения строк, наборов символов).

```
var char: Character = "A" //a это символ
var line: String = "A это строка"
```

Рис 1.8.2 Объявление символа и строки

Забегая вперед, поговорим об интерполяции строк.

Интерполяция строки – это метод, который позволяет вставить в строку другую переменную.

Обратите внимание на пример ниже (Рис 1.8.3). В конце строки после двоеточия мы добавили символ переменной char при помощи конструкции "\(имя_переменной)". Таким образом мы можем добавлять в текст строки и контейнеры других типов данных.

```
var char: Character = "A"
var line: String = "Здесь будет символ: \(char)"
```

Рис 1.8.2 Пример интерполяции символа и строки

 **Домашнее задание!**

Создайте проект Playground. Создайте переменные каждого типа (Char и String). Ответьте на вопрос: "Что такое интерполяция строки?".

Урок 9
Логический тип данных. Логические операторы.

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/QuZvLwLfZ90

Логический тип данных – тип, который может принимать всего лишь 2 состояния: истину (1) или ложь (0).

В языке Swift такой тип данных носит название Bool, по фамилии известного английского математика и логика Джорджа Буля.

```
var flag: Bool = true
```

Рис 1.9.1 Пример объявления переменной типа Bool

Казалось бы что все совсем просто (всего 2 состояния), но это только корни данной темы. Существует набор логических операций над такими переменными:

логическое НЕ	!
логическое И	&&
логическое ИЛИ	||

Рассмотрим данные операции на примерах. Начнем с операции логического “НЕ”. Значение переменной принимает противоположное значение.

```
var flag: Bool = true
//логическое НЕ (!)
flag = !flag
```

true

false

Рис 1.9.1 Операция “НЕ”

Операция логического “И” несколько сложнее. Но придется заучить эти возможные варианты.

```
//логическое И (&&)
false && false          false
false && true           false
true && false           false
true && true            true
```

Рис 1.9.2 Пример объявления переменной типа Bool

Операция логического “ИЛИ” аналогична. Но следует изучить и эти комбинации.

```
//логическое ИЛИ (||)
false || false          false
false || true           true
true || false           true
true || true            true
```

Рис 1.9.3 Пример объявления переменной типа Bool

В дальнейшем мы немного углубимся в Алгебру логики и изучим основные формулы.

 **Домашнее задание!**

Создайте проект Playground. Создайте переменные a = true, b = true и c = false.

Проанализируйте, чему будет равно данное выражение.

a&&(b||c&&(!c))

Повторите его в среде Xcode и сверьте ответы.

Урок 10
Булева Алгебра. Булевы функции.

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/zUhkqZ_GwaU

Булевой алгеброй называют некое множество A с тремя операциями (конъюнкция, дизъюнкция и отрицание) над двумя элементами: 0 (false) и 1 (true).

Выделяют следующие аксиомы:

a ∨ (b ∨ c) = (a ∨ b) ∨ c	a ∧ (b ∧ c) = (a ∧ b) ∧ c	ассоциативность
a ∨ b = b ∨ a	a ∧ b = b ∧ a	коммутативность
a ∨ (a ∨ b) = a	a ∧ (a ∨ b) = a	законы поглощения
a ∨ (b ∨ c) = (a ∨ b) (a ∨ c)	a ∧ (b ∨ c) = (a ∧ b) (a ∧ c)	дистрибутивность
a ∨ ¬a = 1	a ∧ ¬a = 0	дополнительность
¬(a ∨ b) = ¬a ∧ ¬b	¬(a ∧ b) = ¬a ∨ ¬b	законы де Моргана
a ∨ (a ∧ b) = a	a ∧ (a ∨ b) = a	законы поглощения
a ∨ (¬a ∧ b) = a ∨ b	a ∧ (¬a ∨ b) = a ∧ b	Блейка-Порецкого
a ∨ a = a	a ∧ b = a	идемпотентность
¬¬a = a		закон снятия двойного отрицания
a ∨ 0 = a	a ∧ 1 = a	свойства констант
a ∨ 1 = 1	a ∧ 0 = 0	
¬0 = 1	¬1 = 0	
(a ∨ b) ∧ (¬a ∨ b) = b	(a ∧ b) ∨ (¬a ∧ b) = b	склеивание

В языке Swift принятые знаки логических операций заменяются следующим образом:

¬ на !
V на ||
Λ на &&

Знать наизусть предложенную таблицу аксиом совсем не обязательно, но, если вы запомните хотя бы какую-то часть из них – лишним совсем не будет.

 **Домашнее задание!**

Создайте проект Playground. Создайте переменные a, b и c типа bool. Проверьте несколько из предложенных аксиом, меняя значения переменных с true на false и обратно.

```
//Коммутативность
//a||b = b||a      a&&b == b&&a

a||b                                                    true
b||a                                                    true
a&&b                                                    false
b&&a                                                    false
```

Рис 1.10.1 Пример проверки коммутативности

```
//Склеивание
//(a||b)&&(!a||b) = b      (a&&b)||(!a&&b) = b

a                                                       true
b                                                       false

(a||b)&&(!a||b)                                         false
(a&&b)||(!a&&b)                                         false
```

Рис 1.10.2 Пример проверки склеивания

Урок 11
Операторы сравнения

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/gt1vl1EJgnA

При решении различных задач возникает потребность в сравнении чего либо, например, чисел. Давайте представим ситуацию, что у нас есть две переменных a и b типа Int. В каждой записано значение, но какое мы не знаем, но мы хотим узнать, какая из переменных хранит большее значение. Для решения этой задачи мы воспользуемся одним из шести операторов сравнения.

```
var a = 45
var b = 23

a>b
```

```
45
23


true
```

Рис 1.11.1 Сравнение двух чисел

Результатом данной операции будет либо true (если сравнение истинно, а действительно больше b), либо false (а не больше b).

Возможные операторы сравнения:

==	бинарный оператор сравнения (если слева и справа равные значения, вернет true)
!=	бинарный оператор неэквивалентности (если слева и справа неравные значения, вернет true)
>	бинарный оператор “больше” (если число слева больше чем число справа, вернет true)
<	бинарный оператор “меньше” (если число слева меньше чем число справа, вернет true)
>=	бинарный оператор “больше либо равно” (если число слева больше чем число справа либо они равны, вернет true)

<=	бинарный оператор “меньше либо равно” (если число слева меньше чем число справа либо они равны, вернет true)

Во всех иных случаях операторы вернут false.

 **Домашнее задание!**

Создайте проект Playground. Создайте переменные a, b типа Float. Задайте значения этим переменным на свое усмотрение. Проверьте все шесть операторов сравнения на этих переменных.

Урок 12
Преобразование типов

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/lfQYpOfIK6E

Язык программирования Swift предоставляет широкие возможности для работы с различными типами данных.

Преобразование типов – это процесс изменения контейнером данных (переменной либо константой) типа без потери значения.

Допустим, мы имеем строку “23”, которая содержит число. С таким числом мы пока не можем производить математические операции. Первоначально требуется преобразовать в какой-то тип из числовых типов данных на Ваш выбор.

Для преобразования обычно используется следующая запись:

Название_тип(Имя_преобразуемой_переменной)

```
var line = "23"
var number = Int(line)
```

```
"23"
23
```

Рис 1.12.1 Преобразование из String в Int

На рисунке 1.12.1 приведен пример преобразования значения из типа String в Int.

Домашнее задание!

Создайте проект Playground. Создайте переменную типа Float и равную 34.246 и преобразуйте ее в тип Int. Проанализируйте, что произошло с переменной.

Урок 13
Операторы управления

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/V4aGy_aSVzc

Невозможно представить код программы без операторов управления. В процессе работы программы зачастую требуется в зависимости от определенных данных выполнять одну часть кода, при иных данных – другую.

Для управления выполнения определенных частей кода в языке Swift существует 7 операторов:

- assert
- if
- switch
- while
- repeat while
- for
- guard

Все эти 7 операторов делятся на 2 категории:

- операторы ветвления
- операторы повторения

Операторы ветвления позволяют определить порядок выполнения блоков кода. Операторы повторения позволяют циклически выполнять участок кода заданное число раз.

Урок 14
Утверждения

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/6bzkBdhYgzA

Абсолютно любой программист сталкивается с ситуациями, когда необходимо в программу закладывать логику. Например, если пользователь нажал определенную кнопку, необходимо выполнить одни участок кода, если другую кнопку – другой.

Все операторы управления в языке Swift можно разделить на две больших группы:

- Операторы ветвления (Позволяют выбирать, какой участок кода выполнять);
- Операторы повторения (Позволяют многократно выполнять один и тот же участок кода).

К операторам ветвления относятся и утверждения. Утверждения позволяют прерывать выполнение программы, если какое-либо не выполняется (возвращает false). В языке Swift утверждения обозначаются функцией assert(). Существуют две формы использования данного оператора. Простая форма и с возвращаемым отладочным сообщением.

```
assert(_ condition:)
assert(_ condition:, _ message:)
```

Рис 1.14.1 Варианты использования утверждений

Домашнее задание!

Создайте проект Playground. Напишите программу, которая выводит при помощи функции print() текст на экран. Расположите утверждение перед данной функцией и, изменяя проверяемое выражение, проследите за поведением программы.

Урок 15
Оператор ветвления if

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/FyckLIMpIuw

Наверно, ни одна программа не обойдется без оператора ветвления if. Данный оператор как бы разделяет код на две части. Первую часть кода он выполнит при условии, что проверяемое выражение вернет true. Вторую часть, проверяемое выражение вернет false.

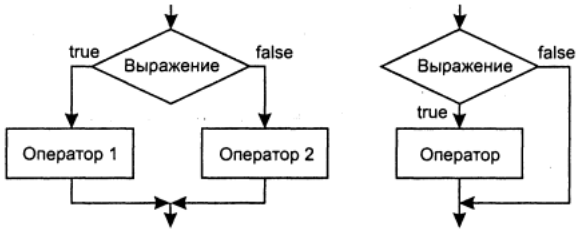


Рис 1.15.1 Блок-схема оператора if

Существуют четыре формы записи оператора условия if, различающихся по синтаксису и функциональному назначению:

- сокращенная
- стандартная
- расширенная
- тернарная

Сокращенная форма позволяет проверить на истинность выражение. Если результат равен true, блок кода, заключенный в фигурные скобки, выполнится, иначе просто будет пропущен.

```
var flag=true
if flag {
    //тело оператора
}
```

Рис 1.15.2 Сокращенная форма оператора if

Стандартная форма дополняет сокращенную форму и позволяет в случае, если проверяемое выражение равно false выполнить второй блок кода после else.

```
if flag {
    //первый блок кода
    //выпоняется при
    //flag = true
} else {
    //второй блок кода
    //выпоняется при
    //flag = false
}
```

Рис 1.15.3 Стандартная форма оператора if

Расширенная форма позволяет добавить проверку еще одного условия.

```
if flag1 {
    //первый блок кода
    //выпоняется при
    //flag = true
} else if flag2 {
    //второй блок кода
    //выпоняется при
    //flag1 = false и
    //flag2 = true
}
else {
    //третий блок кода
    //выпоняется при
    //flag1 = false и
    //flag2 = false
}
```

Рис 1.15.4 Расширенная форма оператора if

Тернарный оператор достаточно сильно отличается своим синтаксисом, но прост в понимании. Первоначально следует проверяемое выражение, затем знак вопроса и два блока кода,

разделенных двоеточием. Левый блок исполняется если проверяемое выражение истинно, правый – ложно.

```
flag ? print("flag = true") : print("flag = false")
```

Рис 1.15.5 Тернарная форма оператора if

Домашнее задание!

Создайте проект Playground. Напишите программу, которая позволяет вычислить корни квадратного уравнения. Если необходимо, обратитесь к математической литературе.

Урок 16
Оператор ветвления switch

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/nwLfhkkKcGM

Достаточно часто возникают ситуации, когда приходится делать выбор в зависимости от значения переменной либо константы. Переменная может принимать набор определённый набор значений. Нам необходимо проверять, какое значение приняла переменная, и в зависимости от этого выполнять определённый блок кода.

Для выполнения данной задачи подойдёт и условный оператор if, но написание исходного кода значительно усложнится и увеличится в своем объеме. Для того чтобы упростить жизнь разработчикам программ, создатели языка Swift придумали оператор ветвления switch.

Оператор switch состоит из так называемого заголовка, в котором указывается название самого оператора, затем следует название переменной либо константы (проверяемого выражения), а уже затем следует набор кейсов (case). Каждый кейс описывает то значение, которое может принимать проверяемое выражение. И в зависимости от подходящего кейса выполняется блок кода, заключённый внутри него.

```
var number = 5
switch number {
case 1:
    print("Выполнится, если number==1")
case 2:
    print("Выполнится, если number==2")
case 3:
    print("Выполнится, если number==3")
default:
    print("Выполнится, если number не нашлось")
}
```

Рис 1.16.1 Структура оператора switch

Стоит обратить внимание, что не всегда кейсы описывают все возможные значения. В том случае если значение не было указано в кейсе выполняется часть default. В default указывается тот код, который необходимо выполнить при отсутствии проверяемого значения в кейсах. Если нам необходимо проигнорировать такой случай и ничего не делать, в блоке default показывается оператор break.

```
var number = 5
switch number {
case 1:
    print("Выполнится, если number==1")
case 2:
    print("Выполнится, если number==2")
case 3:
    print("Выполнится, если number==3")
default:
    break
}
```

Рис 1.16.2 Структура оператора switch с break

Урок 17
Операторы повторения while и repeat while

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/Q9CK-Oa30BU

В процессе обучения мы познакомились с достаточно большим разнообразием операторов управления. Однако, возникают ситуации, когда необходимо циклически (многократно) выполнять участки одного и того же кода с различными входными параметрами. Для выполнения таких задач нам на помощь приходят операторы while и repeat while.

Оператор while позволяет циклически выполнять блок кода до тех пор, пока результат проверяемого выражения возвращает true. Стоит обратить внимание на то, что проверяется выражение перед выполнением кода. Структура оператора while выглядит следующим образом.

```
var i=1
while i<=5 {
    print(i)
    i+=1
}
```

Рис 1.17.1 Структура оператора while

В данном примере цикл выполняется до тех пор, пока не перестанет выполняться выражение *i<=5*. Причем, каждый раз значение переменной *i* увеличивается на единицу.

Наверно, Вы уже задались вопросом, зачем нужен оператор repeat while и чем он отличается от while. Одним и в то же время главным отличием является отсутствие проверки условия на первой итерации цикла. То есть цикл вне зависимости от значения проверяемого выражения сначала выполнит код, а лишь затем его проверит.

```
i=6
repeat {
    print(i)
    i+=1
} while i<=5
```

Рис 1.17.2 Структура оператора repeat while

В данном примере (Рис 1.17.2) видно, что проверяемое выражение уже вначале возвратит false, и цикл выполняться не будет. Но т.к. это цикл с постусловием (сначала выполняется, а потом проверяется условие), то одна итерация цикла будет выполнена.

Урок 18
Операторы повторения for

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/7mMzLXWo7KM

Оператор for предназначен для циклического выполнения блока кода. Однако он удобен в использовании при работе с множествами, последовательностями или массивами.

Представьте, что у Вас имеется массив чисел [1,4,4,7,9] (массивы мы изучим немного позже). Массив представляет из себя набор чисел. Нам необходимо каждое число забрать из массива и вывести в отладочную печать. На помощь в такой ситуации приходит оператор for.

```
for i in [1,4,4,7,9] {
    print(i)
}
```

Рис 1.18.1 Структура оператора for

На каждой итерации поочередно забирается значение из массива и присваивается переменной *i*. И выполняется блок кода, в нашем случае функция print.

Аналогичным образом можно работать с последовательностями.

```
for i in 1...5 {
    print(i)
}
```

Рис 1.18.2 Оператор for на примере последовательности

Хотелось бы уделить внимание функции *stride(from:to:by:)*, которая позволяет задавать начальное (*from:*), конечное значения (*to:*) входного параметра и то, как он будет изменяться (*by:*).

Обратите внимание, что конечное значение не включается при *stride(from:to:by:)*. Для того чтобы вывести значения по конечное включительно используйте форму *stride(from:through:by:)*.

Пусть нам необходимо вывести все нечетные числа от 1 до 1000. Воспользуемся функцией *stride(from:through:by:)*.

```
for i in stride(from: 1, through: 1000, by: 2) {
    print(i)
}
```

Рис 1.18.3 Оператор for с функцией stride

Мы указали шаг 2 для того, чтобы перескакивать четные значения и выводить только нечетные, начав с 1.

 **Домашнее задание!**

Создайте проект Playground. Аналогичным образом выведите все четные целые числа от 10 до 72.

Урок 19
Кортежи или Tuple

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/6SEZ_rAhYew

Возможно, однажды Вы столкнетесь с такой ситуацией, когда будет необходимо объединить несколько элементов разных типов данных в одно значение. В таком случае хорошо подойдут кортежи.

В языке Swift для обозначения кортежей служат открывающая и закрывающая круглые скобки, внутри которых перечисляются элементы. Эти элементы могут быть как одного типа, так и различных.

```
var varible = (3, "Дмитрий", 4.8)
```

Рис 1.19.1 Объявление кортежа

Каждому элементу кортежа присваивается упорядоченный индекс, начиная с 0. Поэтому Вы легко можете обращаться к элементам кортежа.

```
varible.1    "Дмитрий"
```

Рис 1.19.2 Обращение к элементам кортежа

Существуют и другие способы обращения к элементам кортежа, один из них мы рассмотрим.

```
var (number, name, avg) = varible
print("Номер по журналу: \(number). Имя ученика: \(name). Средний балл: \(avg).")
```

Рис 1.19.3 Извлечение элементов кортежа через присвоение

Через объявление трех переменных мы извлекли значения кортежа и использовали их при выводе сообщения на экран.

Урок 20
Диапазоны или Range

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/MULiMVioRG0

Настало время познакомиться с новыми типами данных – диапазонами. Такие переменные позволяют указывать сразу набор значений, например чисел. Необходимо лишь указать границы.

Различают полуоткрытые и закрытые диапазоны.

Оператор полуоткрытого диапазона имеет бинарную и префиксную формы. Бинарная форма позволяет задать левую и правую границы. Префиксная форма позволяет задать лишь одну границу, поэтому такой диапазон бесконечен. Стоит отметить, у таких диапазонов правая граница не включается.

```
var range = 1..<100
```

Рис 1.20.1 Полуоткрытый диапазон с бинарной формой

```
var range = ..<100
```

Рис 1.20.2 Полуоткрытый диапазон с префиксной формой

Главной отличительной особенностью закрытых диапазонов является возможность захватывать границы (включать в диапазон). Такие диапазоны имеют три формы: бинарную, постфиксную и префиксную. Бинарная форма позволяет указать левую и правую границы, а постфиксная и префиксная указывают значение, от которого другие значения изменяются влево или вправо.

```
var range = 1...100
```

Рис 1.20.3 Закрытый диапазон с бинарной формой

В отличие от бинарной формы в постфиксной форме отсутствует правая граница.

```
var range = 1...
```

Рис 1.20.4 Закрытый диапазон с постфиксной формой

Префиксная форма абсолютно похожа на постфиксную.

```
var range = ...100
```

Рис 1.20.5 Закрытый диапазон с префиксной формой

Урок 21
Массивы

❗ Данный видео-урок расположен по следующему адресу: https://youtu.be/B8fv4-be0hs

Массив – это упорядоченный набор элементов, принадлежащих одному типу данных. Причем каждый элемент имеет свой индекс. Нумерация элементов начинается с 0 и с каждым элементом увеличивается на единицу.

Рис 1.21.1 Структура массива

Тип любого массива определяется следующим выражением Array<T>, где T-тип элементов массива.

```
var array: Array<Int> = []
```

Рис 1.21.2 Инициализация пустого массива

Наполнить массив значениями при инициализации совсем не сложно. Необходимо лишь указать значения внутри квадратных скобок, разделенных запятыми.

```
var array = [5, 10, -8, 14, 1, 0, 5]
```

Рис 1.21.3 Инициализация массива целыми числами

Обратиться к элементам массива совсем не сложно. Для этого необходимо написать название массива, затем в квадратных скобках указать индекс запрашиваемого элемента.

```
array[3]    14
```

Рис 1.21.2 Обращение к элементу массива