POLYTECH
COLLEGE ALMATY

Swift бағдарламалау тілі

Swift

Swift — это новый язык программирования, разработанный Apple Inc. для разработки iOS и OS X. Он использует лучшее из C и Objective-C, без ограничений совместимости с C. Он использует ту же среду выполнения, что и существующая система Obj-C на Mac OS и iOS, что позволяет запускать программы Swift на многих существующих платформах iOS и OS X.

Этот учебник по Swift поможет вам понять Swift очень простым и легким способом. Так что вы сможете создать свое собственное приложение или программу Swift. Он охватит все основные концепции языка программирования Swift, что повысит вашу уверенность и сделает вас хорошим программистом Swift.

Что такое язык программирования Swift?

Swift — это современный язык программирования с открытым исходным кодом, специально разработанный Apple для своих платформ. Он был представлен в 2014 году с целью предоставить язык, который не только мощный и универсальный, но и обеспечивает большую безопасность, производительность, совместимость с Objective-C и современным синтаксисом. Поэтому разработчики Swift могут легко разрабатывать надежные и высокопроизводительные приложения.

The latest version of Swift is Swift 5.9.2

Мы также можем использовать Swift для разработки программного обеспечения для телефонов, настольных компьютеров и серверов. Swift — это великолепное сочетание современного мышления и разнообразных вкладов сообщества разработчиков открытого кода. Компилятор Swift оптимизирован для своей производительности, а сам язык адаптирован для своей разработки.

Зачем изучать Swift?

Если вы заинтересованы в разработке приложений для экосистемы Apple, то Swift для вас. Swift открывает вам шлюз для создания динамичных, инновационных и мощных приложений для iOS, macOS, watchOS и tvOS. Apple предпочитает язык программирования Swift в качестве основного языка, поскольку он имеет современный синтаксис, высокую производительность, обеспечивает большую безопасность и без проблем работает со всеми устройствами.

Мы также можем использовать Swift для создания приложений для Windows и Android благодаря его открытому исходному коду и кроссплатформенной совместимости. Кроссплатформенная разработка приложений позволяет разработчикам писать код, а затем развертывать его на нескольких платформах, например, Flutter, React Native и Xamarin. Несмотря на то, что Swift предлагает эту универсальность, пользовательский опыт может быть не таким бесшовным, как при использовании других языков программирования, таких как C#, .Net, Java, Kotlin и т. д.

Быстрые вакансии и возможности

В динамичную эру технологий язык программирования Swift стал важной вехой в создании удобных для пользователя приложений для продуктов Apple. Спрос на экспертные знания Swift достигает новых высот, и рынок переполнен множеством возможностей для талантливых разработчиков. Независимо от того, являетесь ли вы опытным разработчиком Swift или новичком, у вас будет много возможностей с хорошими пакетами. Средняя зарплата разработчика Swift составляет от 5 до 12 фунтов в год, она может варьироваться в зависимости от местоположения, должности и опыта.

Есть так много компаний, которые предоставляют хороший пакет и рабочую культуру для разработчиков Swift. Невозможно перечислить все названия компаний, которые используют Swift, но некоторые

- Яблоко
- Google
- Фейсбук
- Майкрософт
- Амазонка
- Твиттер
- Airbnb
- Снэпчат
- Adobe
- Пинтерест
- Слак

- Убер
- Нетфликс

Онлайн-компилятор Swift

Мы предоставили **онлайн-компилятор/интерпретатор Swift** , который поможет вам редактировать и выполнять код прямо из браузера.

Пример

Открытый компилятор

```
// First Swift program
print("Hello! Swift")
```

Выход

```
Hello! Swift
```

Карьера в Swift

Swift — мощный и интуитивно понятный язык для разработки программного обеспечения. Он обеспечивает надежную платформу для создания динамичных и эффективных приложений для экосистемы Apple. Он обычно используется для создания бесшовных и инновационных приложений для iOS, macOS, watchOS и tvOS. Swift открывает двери для огромных возможностей, где разработчик может продемонстрировать свои навыки. Ниже приведены некоторые потенциальные варианты карьеры с языком программирования Swift −

- Разработчик приложений для iOS/macOS
- Разработчик мобильных приложений
- Разработчик игр
- Разработчик дополненной реальности (AR)
- UI/UX-дизайнер для iOS-приложений
- Инженер по контролю качества (QA) для приложений iOS
- Разработчик iOS-фреймворка
- Тренер Swift
- Технический писатель Swift
- Кроссплатформенный мобильный разработчик
- Полный стек Swift-разработчик
- Разработчик приложений WatchOS

Кому следует изучать Swift

Это руководство предназначено для программистов, которые хотели бы изучить основы языка программирования Swift с нуля. Это руководство даст вам достаточно знаний о языке программирования Swift, чтобы вы могли перейти на более высокий уровень знаний.

Предпосылки для изучения Swift

Прежде чем приступить к изучению этого руководства, вы должны иметь базовые знания терминологии компьютерного программирования и опыт работы с любым языком программирования.

Часто задаваемые вопросы о Swift

Есть несколько часто задаваемых вопросов (FAQ) о Swift, в этом разделе мы попытаемся кратко на них ответить.

Обзор Свифта

Версия	**Год выпуска**
Свифт 1.0	2014
Свифт 1.2, Свифт 2.0	2015
Свифт 3.0	2016
Свифт 4.0	2017
Свифт 4.1, 4.2	2018
Свифт 5.0, Свифт 5.1	2019

Свифт 5.3	2020
Свифт 5.4, Свифт 5.5	2021
Свифт 5.6, Свифт 5.7	2022
Свифт 5.8, Свифт 5.9	2023

Характеристики Свифта

Swift предлагает своим разработчикам различные символы, которые помогут им разрабатывать оптимизированные и динамичные приложения для платформы Apple, а ключевыми характеристиками являются:

- **Современный синтаксис** − Swift обеспечивает чистый и выразительный синтаксис, чтобы разработчик мог разрабатывать лаконичные и легко читаемые программы. Это делает его более доступным языком как для начинающих, так и для опытных разработчиков.
- **Безопасность** − Он повышает безопасность, удаляя распространенные ошибки и недочеты. Он включает в себя современные методы программирования, что делает его более безопасным и надежным.
- **Производительность** — обеспечивает высокую производительность, как и языки низкого уровня, при этом сохраняя безопасность и выразительность кода.
- **Взаимодействие** − Swift — это совместимый язык. Он без проблем работает с другими языками, такими как Objective-C. Разработчикам разрешено использовать код Swift и Objective-C вместе в одном проекте.
- **Открытый исходный код** . Swift — это язык программирования с открытым исходным кодом, который расширяет возможности совместной работы, инноваций и постоянного совершенствования Swift.
- **Необязательные элементы** − Swift обеспечивает большую поддержку необязательных элементов, чтобы разработчики могли явно представлять отсутствие значения. Он очень хорошо обрабатывает нулевые или неопределенные значения без риска сбоя во время выполнения.
- **Вывод типа** − Swift поддерживает сильную систему типов, но также включает в себя интерфейс типов, чтобы снизить потребность в явной аннотации типов. Это означает, что компилятор Swift способен анализировать переменные и выражения и определять их тип.
- **Автоматическое управление памятью** − Swift использует автоматический подсчет ссылок для управления памятью. Он обрабатывает выделение и освобождение памяти автоматически без каких-либо задержек. Поскольку выделение памяти вручную является распространенным источником ошибок для разработчиков.
- **Замыкания** − Swift обеспечивает большую поддержку замыканий. Замыкание — это блочный код, на который можно ссылаться, передавать и выполнять позже. Они повышают модульность и гибкость программы.
- **Протокольно-ориентированное программирование** − Swift поощряет протокольно-ориентированное программирование. Он подчеркивает использование протоколов для создания чертежей функциональности. Он создает повторно используемые, модульные и компонуемые коды.

Применение Свифта

Swift в основном используется для разработки приложений для платформы Apple, но его также можно использовать для разработки приложений для других платформ. Хотя Swift имеет многочисленные приложения, которые невозможно перечислить, поэтому мы приводим некоторые из основных приложений Swift −

- **Разработка приложений для iOS** — это наиболее предпочтительный язык для разработки приложений для устройств iOS, таких как iPad, iPhone и т. д.
- **Разработка для macOS** — также используется для создания приложений, утилит и программного обеспечения для операционной системы macOS.
- **Разработка приложений watchOS** . Используя Swift, мы также можем создавать широкий спектр приложений для здоровья для Apple Watch.
- **Разработка приложений для tvOS** . С помощью Swift мы также можем создавать различные развлекательные приложения для tvOS.

- **Кроссплатформенная разработка** − Swift не ограничивается платформой Apple, мы также можем создавать приложения для других платформ с помощью кроссплатформенной разработки. Например, фреймворк SwiftUI используется для создания приложений или интерфейсов, которые работают как на Windows, так и на iOS.
- **Разработка на стороне сервера** − Swift также используется в разработке на стороне сервера. Он позволяет разработчикам разрабатывать веб-приложения, API и сервисы с использованием серверных фреймворков Swift, таких как Vapour, Kitura и Perfect.

Недостатки Swift

Каждый язык программирования имеет свои плюсы и минусы. Аналогично, Swift также имеет различные преимущества и недостатки. Итак, некоторые из основных недостатков Swift:

- Swift в первую очередь разработан для создания приложений для устройств Apple. Поэтому за пределами экосистемы Apple он имеет ограниченное применение, поскольку у разработчиков есть много альтернативных вариантов для кроссплатформенного языка.
- Swift — новый язык программирования, поэтому в нем нет таких развитых инструментов, как в других языках программирования.
- По сравнению с Java, JavaScript или Python у него небольшой пул разработчиков.
- Совместимость с платформами сторонних производителей ограничена.
- Swift поддерживает вывод типов и оптимизацию, поэтому если проект большой, то время компиляции увеличивается.

Свифт - Окружающая среда

Swift — язык нового поколения, разработанный Apple для создания приложений и программного обеспечения для экосистемы Apple. Благодаря своей скорости, безопасности и современному синтаксису он быстро стал предпочтительным языком среди разработчиков. Первоначально Siwft был разработан для создания приложений для iOS и macOS.

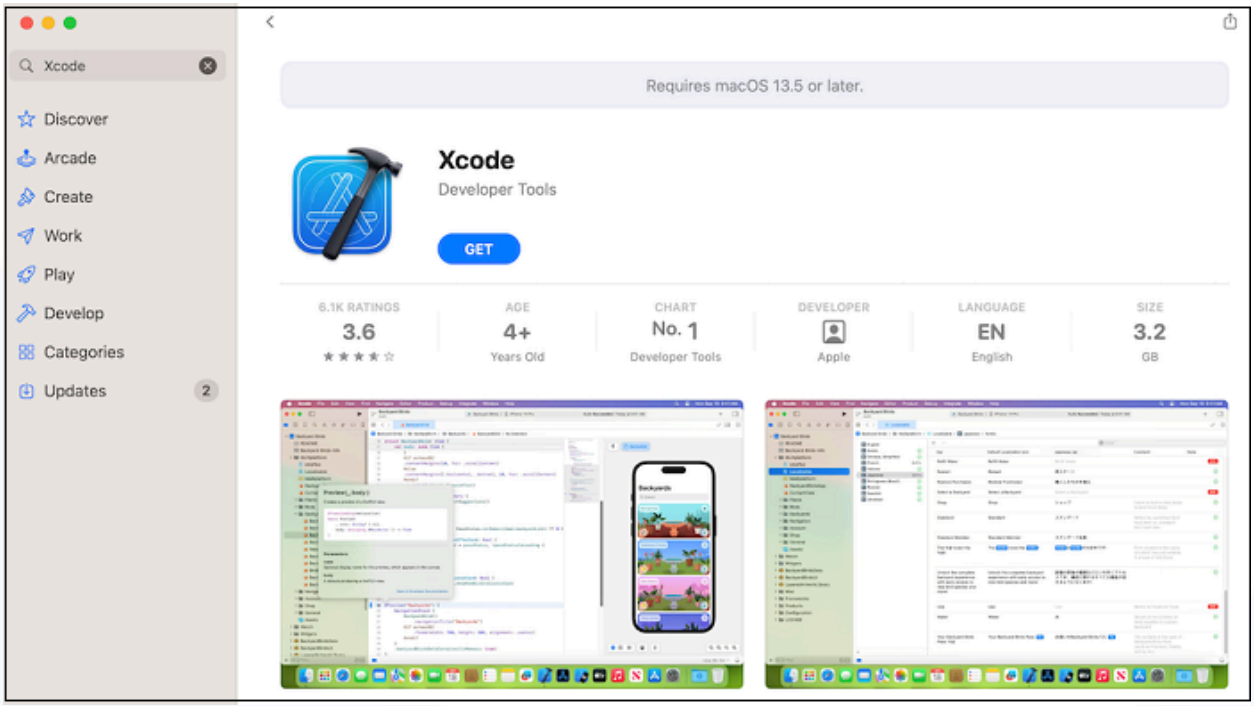
Однако с ростом его популярности разработчики расширили его универсальность. Теперь Swift доступен не только для macOS, но и для Windows и Linux, и предоставляет разработчикам кроссплатформенные решения. Теперь давайте рассмотрим, как можно установить Swift на macOS и Windows шаг за шагом.

Установка Swift на macOS

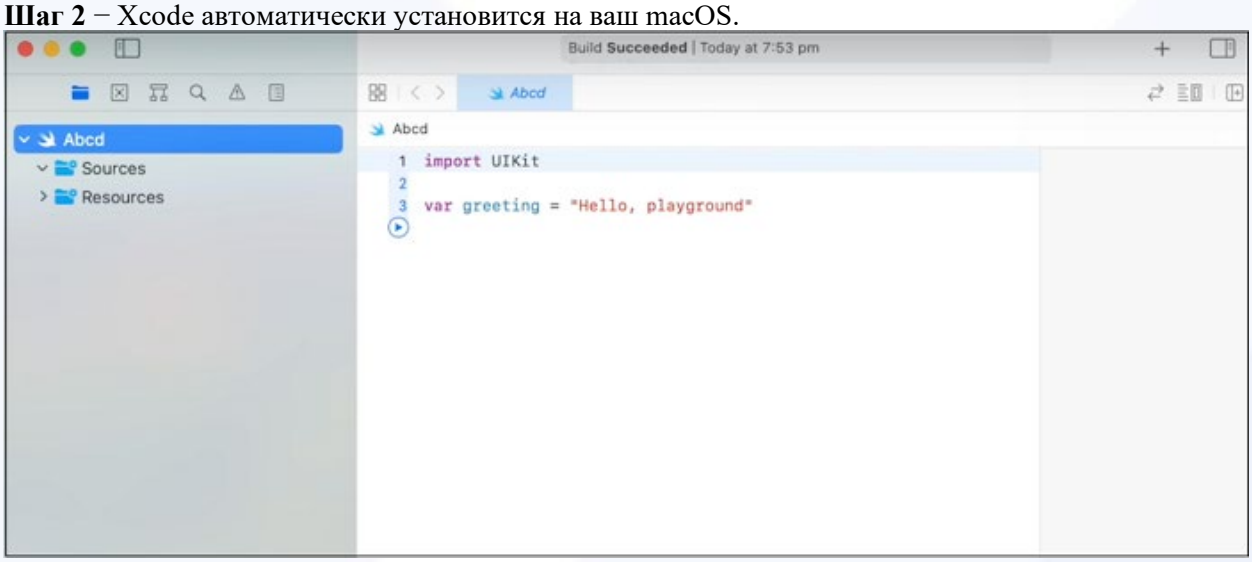
Для установки Swift на macOS нам потребовался Xcode. Xcode — это интегрированная среда разработки (IDE) Apple для всех платформ Apple. Она предоставляет различные инструменты для создания приложений и поддерживает исходный код для различных языков программирования, таких как Swift, Objective-C, Java, Python, C++, C и т. д.

Итак, выполните следующие шаги, чтобы загрузить Xcode в macOS:

Шаг 1 − Перейдите в App Store или перейдите по следующей ссылке, чтобы загрузить последнюю версию Xcode: www.swift.org/install/ .

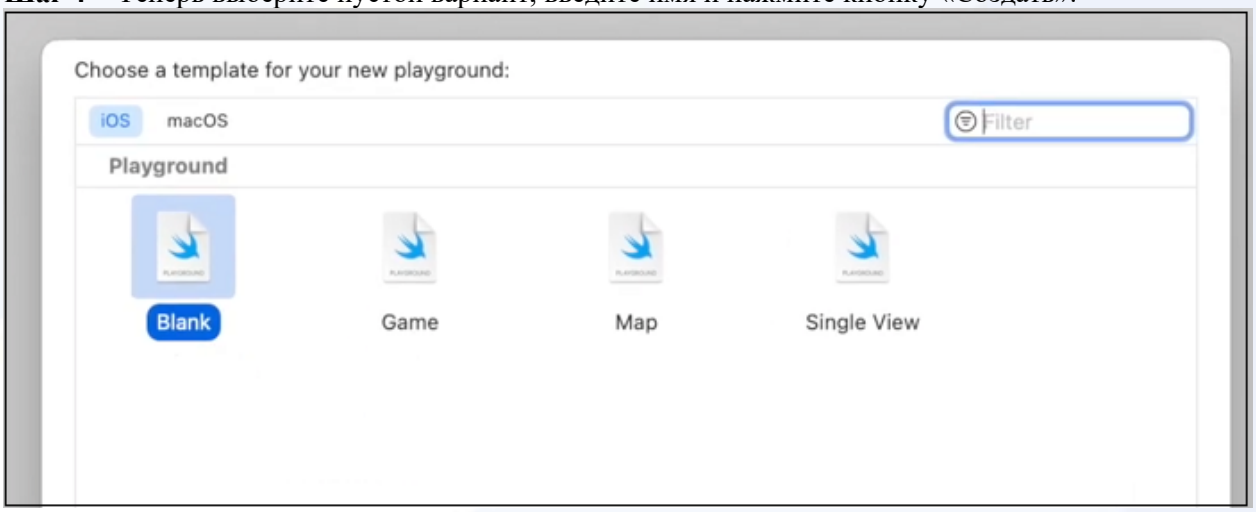


Шаг 2 − Xcode автоматически установится на ваш macOS.



Шаг 3 − Теперь откройте Xcode и перейдите в меню Файл >> Создать >> Игровая площадка.

Шаг 4 − Теперь выберите пустой вариант, введите имя и нажмите кнопку «Создать».

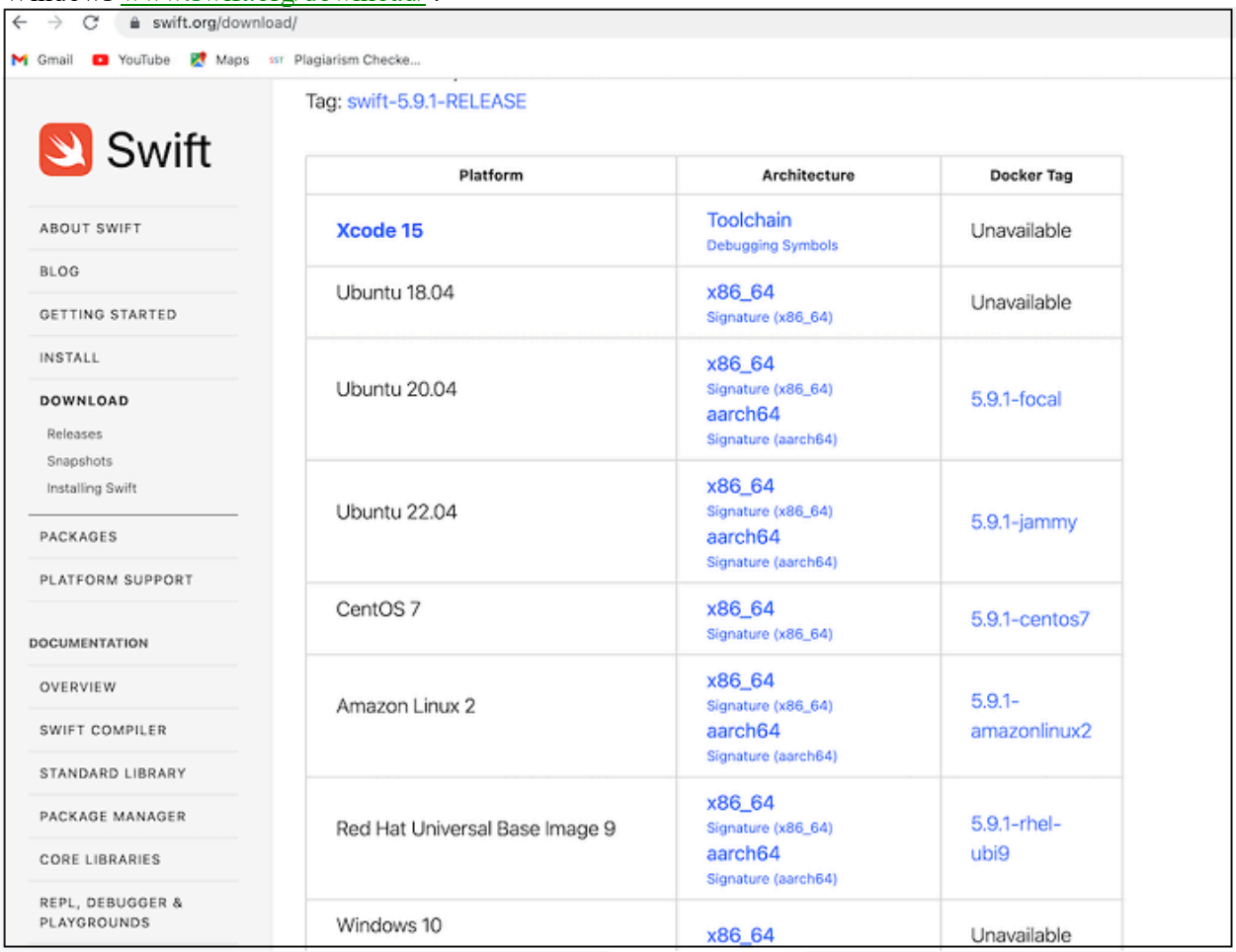


Шаг 5 − Теперь вы готовы написать и запустить свою первую программу на Swift.

Установка Swift на Windows

Чтобы установить Swift на Windows, выполните следующие действия:

Шаг 1 − Перейдите по следующей ссылке, чтобы загрузить последнюю версию Swift для Windows www.swift.org/download/ .

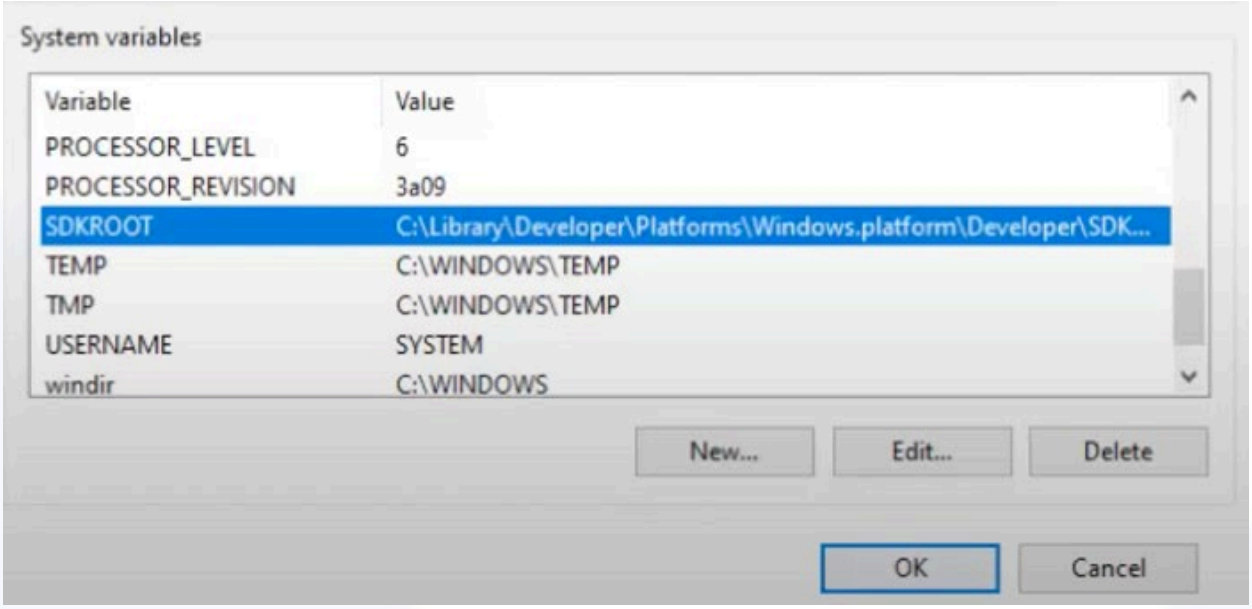


Шаг 2 − Нажмите на Windows x86_64, и начнется загрузка.

Шаг 3 − Перейдите в папку загрузки и откройте загруженный файл .exe.

Шаг 4 − Нажмите кнопку «Установить», и процесс установки начнется.

Шаг 5 − После завершения установки перейдите к переменной среды и проверьте наличие SDKRoot.



Шаг 6 − Теперь перейдите в командную строку и введите следующую команду −

```
Swift —version
```

Распечатать страницу

Swift — базовый синтаксис

Если Swift установлен успешно, он покажет последнюю версию Swift. Swift — мощный и выразительный язык для разработки приложений для устройств Apple. Он разработан так, чтобы его было легко читать, писать и поддерживать. Он также поддерживает объектно-ориентированные и функциональные парадигмы программирования.

Синтаксис

Его синтаксис гораздо более понятен, лаконичен и удобен для чтения. Теперь давайте рассмотрим базовую структуру программы Swift, чтобы вам было легче понять основные строительные блоки языка программирования Swift.

Открытый компилятор

```
/* My first Swift program */
var myString = "Hello, World!"

print(myString)
```

Ниже приведен вывод вышеуказанной программы:

Hello, World!

Теперь давайте узнаем, каковы основные синтаксисы и как их можно использовать в вашей программе Swift.

Импортное заявление

Вы можете использовать оператор **import** для импорта любого фреймворка Objective-C или библиотеки C или библиотеки Swift непосредственно в вашу программу Swift. Например, оператор import cocoa делает все библиотеки Cocoa, API и среды выполнения, которые формируют уровень разработки для всей OS X, доступными в Swift. Cocoa реализован в Objective-C, который является надмножеством C, поэтому легко смешивать C и даже C++ в ваших приложениях Swift.

Синтаксис

Ниже приведен синтаксис оператора импорта:

```
import frameworkName or LibraryName
```

Пример

```
import Foundation
```

Жетоны

Программа Swift состоит из различных токенов. Токен — это наименьшая единица программы, которая используется для построения блоков программ Swift. Они могут быть ключевым словом, идентификатором, константой, строковым литералом или символом.

Пример

Программа Swift для демонстрации использования токенов. Здесь программа содержит 10 токенов, таких как import, Foundation, var, myString и т. д.

Открытый компилятор

```
import Foundation
var myString = 34
print(myString)
```

Выход

34

Точки с запятой

В Swift точки с запятой после каждого оператора необязательны. Это полностью ваш выбор, хотите ли вы ставить точку с запятой (;) после каждого оператора в вашем коде, компилятор не жалуется на это.

Однако если вы используете несколько операторов в одной строке, то необходимо использовать точку с запятой в качестве разделителя, в противном случае компилятор выдаст синтаксическую ошибку.

Пример

Программа на Swift, демонстрирующая использование точек с запятой в одной строке.

Открытый компилятор

```
// Separating multiple statements using semicolon
var myString = "Hello, World!"; print(myString)
```

Выход

Hello, World!

Идентификаторы

Идентификатор Swift — это имя, используемое для идентификации переменной, функции или любого другого определяемого пользователем элемента. Имя идентификатора должно начинаться с буквы алфавита от A до Z или от a до z или с подчеркивания _, за которым следует ноль или более букв, подчеркиваний и цифр (от 0 до 9).

Swift не допускает использования специальных символов, таких как @, $ и %, в идентификаторах. Swift — язык программирования, **чувствительный к регистру** , поэтому Manpower и manpower — это два разных идентификатора в Swift.

Пример

Ниже приведены некоторые допустимые идентификаторы в Swift:

```
Azad      zara    abc   move_name  a_123
myname50  _temp   j     a23b9      retVal
```

Ключевые слова

Ключевые слова — это специальные предопределенные слова, доступные в Swift, которые имеют особое значение и функциональность. Они также известны как зарезервированные слова. Эти зарезервированные слова не могут использоваться как константы, переменные или любые другие имена идентификаторов, если они не экранированы обратными кавычками (`).

Например, class не является допустимым идентификатором, но `class` допустим. Они обычно предназначены для определения структуры и поведения программ. Swift поддерживает следующие ключевые слова −

Ключевые слова, используемые в декларациях

Ниже приведены ключевые слова, используемые в декларациях.

Сорт	деинит	Перечисление	расширение
Функц	импорт	Инициализация	внутренний
Позволять	оператор	частный	протокол
публичный	статический	структура	нижний индекс

typealias	вар

Ключевые слова, используемые в заявлениях
Ниже приведены ключевые слова, используемые в заявлениях.

перерыв	случай	продолжать	по умолчанию
делать	еще	провал	для
если	в	возвращаться	выключатель
где	пока		

Ключевые слова, используемые в выражениях и типах
Ниже приведены ключевые слова, используемые в выражениях и типах.

как	динамическийТип	ЛОЖЬ	является
ноль	себя	Себя	супер
истинный	_СТОЛБЕЦ_	_ФАЙЛ_	_ФУНКЦИЯ_
ЛИНИЯ			

Ключевые слова, используемые в определенных контекстах
Ниже приведены ключевые слова, используемые в определенных контекстах.

ассоциативность	удобство	динамический	didSet
финал	получать	инфикс	внут.
ленивый	левый	мутирующий	никто
немутирующий	необязательный	переопределить	постфикс
приоритет	префикс	Протокол	необходимый
верно	набор	Тип	бесхозный
слабый	willSet		

Пробелы
Пробел — это термин, используемый в Swift для описания пробелов, табуляции, символов новой строки и комментариев. Пробелы отделяют одну часть оператора от другой и позволяют компилятору определить, где заканчивается один элемент в операторе, например int, и начинается следующий элемент. Поэтому в следующем операторе −

```
var age
```

Между var и **age** должен быть хотя бы один пробел (обычно пробел), чтобы компилятор мог их различить. С другой стороны, в следующем операторе:

```
//Get the total fruits
int fruit = apples + oranges
```

Между фруктами и = или между = и яблоками не требуется никаких пробелов, хотя вы можете использовать их для лучшей читаемости.
Пространство по обе стороны от оператора должно быть одинаковым, например −

```
int fruit= apples +oranges   //is a wrong statement
int fruit = apples + oranges   //is a Correct statement
```

Литералы

Литерал — это представление исходного кода значения целого числа, числа с плавающей точкой или строкового типа. Или можно сказать, что литералы — это фактические значения, которые

напрямую записаны в исходном коде. Swift поддерживает целочисленные литералы, литералы с плавающей точкой, строковые литералы, булевы литералы и т. д.

Пример

Ниже приведены примеры литералов:

```
92                  // Integer literal
4.24159             // Floating-point literal
"Hello, World!"     // String literal
```

Печать в Swift

В Swift вы можете вывести что угодно на консоль с помощью функции print(). Это самая полезная функция в языке Swift. Она также может выводить файлы, функции и номера строк.

Синтаксис

Ниже приведен синтаксис функции print():

```
func print(_ element: Any…, Sep: String = “ “, Terminator: String = “\n”)
```

Параметры

Эта функция принимает только три параметра, и они необязательны:

- **Элемент** − Это элемент, который мы хотим вывести в консоль. Он может быть равен нулю или больше.
- **Sep** − представляет строку, которая будет отображаться между каждым элементом. По умолчанию занимает один пробел.
- **Терминатор** − представляет собой строку, которая будет выведена после всех элементов. По умолчанию ее значение — новая строка. Мы можем задать собственные разделители и терминаторы в соответствии с нашими требованиями.

Пример

Программа на языке Swift, демонстрирующая, как печатать текст с помощью функции **print()** .

Открытый компилятор

```
import Foundation

// Printing text
print("Learn Swift!")

// Printing variables
var x = 10
var y = 23
print("[\(x) and \(y)]")

// Printing text with terminator
print("Swift, Programming Language", terminator: "*")
print("Swift Programming Language")
```

Выход

```
Learn Swift!
[10 and 23]
Swift, Programming Language*Swift Programming Language
```

Распечатать стр.

Swift - Переменные

Что такое переменные в Swift?

Переменная предоставляет нам именованное хранилище, которым могут манипулировать наши программы. Каждая переменная в Swift имеет определенный тип, который определяет размер и структуру памяти переменной. Переменная может хранить диапазон значений в этой памяти или набор операций, которые могут быть применены к переменной.

Переменные являются статическими типами, то есть после объявления их с определенным типом их тип не может быть изменен, может быть изменено только их значение.

Объявление переменных Swift

Объявление переменной сообщает компилятору, где и сколько создать хранилище для переменной. Они всегда объявляются перед их использованием и объявляются с использованием ключевого слова var.

Синтаксис

Ниже приведен синтаксис переменной:

```
var variableName = value
```

Пример

Программа на языке Swift, демонстрирующая, как объявлять переменные.

```
Открытый компилятор
import Foundation

// Declaring variable
let number = 42
print("Variable:", number)
```

Выход

Variable: 42

Мы также можем объявить несколько переменных в одной строке. Где каждая переменная имеет свои значения и разделена запятыми.

Синтаксис

Ниже приведен синтаксис нескольких переменных:

```
var variableA = value, variableB = value, varaibleC = value
```

Пример

Программа на языке Swift, демонстрирующая, как объявить несколько переменных в одной строке.

```
Открытый компилятор
import Foundation

// Declaring multiple variables
var variableA = 42, variableB = "TutorialsPoint", variableC = 3.3

print("Variable 1:", variableA)
print("Variable 2:", variableB)
print("Variable 3:", variableC)
```

Выход

Variable 1: 42
Variable 2: TutorialsPoint
Variable 3: 3.3

Типовые аннотации в Swift

Аннотация типа используется для определения того, какой тип значения должен храниться в переменной во время объявления. При объявлении переменной мы можем указать аннотацию типа, поместив двоеточие после имени переменной, за которым следует тип.

Аннотирование типа используется редко, если мы указываем начальное значение во время объявления переменной, поскольку Swift автоматически выводит тип переменной в соответствии с присвоенным значением.

Синтаксис

Ниже приведен синтаксис аннотаций типов:

```
var variableName : Type = Value
```

Пример

Программа на языке Swift, демонстрирующая, как задать аннотацию типа.

```
Открытый компилятор
import Foundation

// Declaring variable with type annotation
var myValue : String = "Swift Tutorial"
print("Variable:", myValue)
```

Выход

Variable: Swift Tutorial

Мы также можем определить несколько переменных одного типа в одной строке. Где каждое имя переменной разделяется запятой.

Синтаксис

Ниже приведен синтаксис нескольких переменных:

```
let variableA, variableB, varaibleC : Type
```

Пример

Программа на языке Swift, демонстрирующая, как указать несколько переменных в аннотации одного типа.

Открытый компилятор

```
import Foundation

// Declaring multiple variables in single-type annotation
var myValue1, myValue2, myValue3 : Int

// Assigning values
myValue1 = 23
myValue2 = 22
myValue3 = 11

print("Variable Value 1:", myValue1)
print("Variable Value 2:", myValue2)
print("Variable Value 3:", myValue3)
```

Выход

```
Variable Value 1: 23
Variable Value 2: 22
Variable Value 3: 11
```

Именование переменных в Swift

Присвоение имени переменной — очень важная часть во время объявления. Они должны иметь уникальное имя. Вам не разрешено хранить две переменные с одинаковым именем, если вы попытаетесь это сделать, вы получите ошибку. Swift предоставляет следующие правила для присвоения имени переменной −

- Имена переменных могут содержать любые символы, включая символы Юникода. Например, var 你好 = "你好世界".
- Имя переменной не должно содержать пробелов, математических символов, стрелок, закрытых скалярных значений Unicode, а также символов рисования линий и рамок.
- Имя переменной может состоять из букв, цифр и символа подчеркивания. Оно должно начинаться либо с буквы, либо с символа подчеркивания. Например, var myValue = 34.
- Заглавные и строчные буквы различаются, поскольку Swift чувствителен к регистру. Например, var value и var Value — это две разные переменные.
- Имена переменных не должны начинаться с цифры.
- Вам не разрешено повторно объявлять переменную с тем же именем. Или не разрешается изменять ее тип на другой.
- Вам не разрешается изменять переменную на константу и наоборот.
- Если вы хотите объявить имя переменной таким же, как зарезервированное ключевое слово, то используйте обратные кавычки (') перед именем переменной. Например, var ' **break = "hello"** .

Печать переменной Swift

Вы можете распечатать текущее значение переменной с помощью функции **print()** . Вы можете интерполировать значение переменной, заключив имя в скобки и экранировав его обратной косой чертой перед открывающейся скобкой.

Пример

Программа Swift для печати переменных.

Открытый компилятор

```
import Foundation

// Declaring variables
var varA = "Godzilla"
var varB = 1000.00

// Displaying constant
print("Value of \(varA) is more than \(varB) million")
```

Выход
Value of Godzilla is more than 1000.0 million

Swift - Константы

Что такое Constant в Swift?
Константы относятся к фиксированным значениям, которые программа не может изменять во время выполнения. Константы могут быть любого типа данных, например, целые числа, числа с плавающей точкой, символьные, двойные или строковые литералы. Существуют также константы перечисления. Они обрабатываются так же, как обычные переменные, за исключением того, что их значения не могут быть изменены после их определения.
Объявление константы Swift
Константы используются для хранения значений, которые не будут меняться на протяжении всего выполнения программы. Они всегда объявляются перед использованием и объявляются с использованием ключевого слова let.
Синтаксис
Ниже приведен синтаксис константы:

```
let number = 10
```

Пример
Программа на языке Swift, демонстрирующая, как объявлять константы.

```
Открытый компилятор
import Foundation

// Declaring constant
let constA = 42
print("Constant:", constA)
```

Выход
Constant: 42
Если мы попытаемся присвоить значение константе, то получим ошибку, как в примере ниже:

```
Открытый компилятор
import Foundation

// Declaring constant
let constA = 42
print("Constant:", constA)

// Assigning value to a constant
constA = 43
print("Constant:", constA)
```

Выход

```
main.swift:8:1: error: cannot assign to value: 'constA' is a 'let' constant
constA = 43
^~~~~~
main.swift:4:1: note: change 'let' to 'var' to make it mutable
let constA = 42
^~~
var
```

Мы также можем объявить несколько констант в одной строке. Где каждая константа имеет свои значения и разделена запятыми.
Синтаксис
Ниже приведен синтаксис множественных констант:

```
let constantA = value, constantB = value, constantC = value
```

Пример
Программа на языке Swift, демонстрирующая, как объявить несколько констант в одной строке.

```
Открытый компилятор
import Foundation
```

```
// Declaring multiple constants
let constA = 42, constB = 23, constC = 33
print("Constant 1:", constA)
print("Constant 2:", constB)
print("Constant 3:", constC)
```

Выход

Constant 1: 42
Constant 2: 23
Constant 3: 33

Тип Аннотации с константами

Аннотация типа используется для определения того, какой тип значения должен храниться в константе во время объявления. При объявлении константы мы можем указать аннотацию типа, поместив двоеточие после имени константы, за которым следует тип.

Аннотирование типа используется редко, если мы указываем начальное значение во время объявления константы, поскольку Swift автоматически выводит тип константы в соответствии с назначенным значением.

Например, пусть **myValue** = **"hello"** , тогда тип константы myValue — String, поскольку мы присвоили константе строковое значение.

Синтаксис

Ниже приведен синтаксис аннотаций типов:

```
let constantName : Type = Value
```

Пример

Программа на языке Swift, демонстрирующая, как задать аннотацию типа.

Открытый компилятор

```
import Foundation

// Declaring constant with type annotation
let myValue : String = "hello"
print("Constant:", myValue)
```

Выход

Constant: hello

Мы также можем определить несколько констант одного типа в одной строке. Где каждое имя константы разделяется запятой.

Синтаксис

Ниже приведен синтаксис множественных констант:

```
let constantA, constantB, constantC : Type
```

Пример

Программа на Swift, демонстрирующая, как указать несколько констант в аннотации одного типа.

Открытый компилятор

```
import Foundation

// Declaring multiple constants in single-type annotation
let myValue1, myValue2, myValue3 : String

// Assigning values
myValue1 = "Hello"
myValue2 = "Tutorials"
myValue3 = "point"

print("Constant Value 1:", myValue1)
print("Constant Value 2:", myValue2)
print("Constant Value 3:", myValue3)
```

Выход

Constant Value 1: Hello
Constant Value 2: Tutorials
Constant Value 3: point

Именование констант в Swift

Именование константы очень важно. Они должны иметь уникальное имя. Вам не разрешено хранить две константы с одинаковым именем, если вы попытаетесь это сделать, вы получите ошибку. Swift предоставляет следующие правила для именования константы −

- Имена констант могут содержать любые символы, включая символы Юникода. Например, пусть 你好 = «你好世界».
- Имя константы не должно содержать пробелов, математических символов, стрелок, закрытых скалярных значений Unicode или символов рисования линий и рамок.
- Имя константы может состоять из букв, цифр и символа подчеркивания. Оно должно начинаться либо с буквы, либо с символа подчеркивания. Например, пусть myValue = 34.
- Прописные и строчные буквы различаются, поскольку Swift чувствителен к регистру. Например, let value и let Value — это две разные константы.
- Имена констант не должны начинаться с цифры.
- Вам не разрешено повторно объявлять константу с тем же именем. Или не может измениться на другой тип.
- Вам не разрешается изменять константу в переменную и наоборот.
- Если вы хотите объявить имя константы таким же, как зарезервированное ключевое слово, то используйте обратные кавычки (`) перед именем константы. Например, пусть 'var = “hello”.

Печать констант Swift

Вы можете распечатать текущее значение константы или переменной с помощью функции **print()** . Вы можете интерполировать значение переменной, заключив имя в скобки и экранировав его обратной косой чертой перед открывающейся скобкой.

Пример

Программа Swift для печати констант.

Открытый компилятор

```
import Foundation

// Declaring constants
let constA = "Godzilla"
let constB = 1000.00

// Displaying constant
print("Value of \(constA) is more than \(constB) millions")
```

Выход

Value of Godzilla is more than 1000.0 millions

Свифт - Литералы

Что такое литералы?

Литерал — это представление исходного кода значения целого числа, числа с плавающей точкой или строкового типа. Или можно сказать, что литералы используются для представления фактических значений, которые используются для назначения константы или переменной. Например, 34 — это целочисленный литерал, 23.45 — это литерал с плавающей точкой. Они напрямую используются в программе. Мы не можем напрямую выполнять операции с литералами, поскольку они являются фиксированными значениями, но мы можем выполнять операции с переменными или константами (инициализированными с помощью литералов).

Тип литералов Swift

Swift поддерживает следующие типы литералов:

- Целочисленные литералы
- Литералы с плавающей точкой
- Строковые литералы
- Булевы литералы

Целочисленные литералы Swift

Целочисленный литерал используется для представления целочисленного значения или целого числа. Мы можем указать отрицательное или положительное значение в целочисленном литерале, например, -10 или 10. Целочисленный литерал может содержать ведущие нули, но они не оказывают никакого влияния на значение литерала, например, 010 и 10 оба являются одинаковыми.

Целочисленные литералы бывают следующих типов:

- **Десятичный целый литерал** − это наиболее часто используемая форма целочисленного литерала. Например, 34.
- **Двоичный целый литерал** − используется для представления оснований 2 или двоичных значений. Он имеет префикс **«0b»** . Например, 0b1011.
- **Восьмеричный целый литерал** − используется для представления восьмеричных значений или значений с основанием 8. Он имеет префикс **«0o»** . Например, 0o53.
- **Шестнадцатеричный целый литерал** − используется для представления шестнадцатеричных или шестнадцатеричных значений. Он имеет префикс **«0x»** . Например, 0xFF.

Пример

Программа Swift для демонстрации целочисленных литералов −

Открытый компилятор

```
import Foundation

// Decimal integer literal
let num1 = 34
print("Decimal Integer: \(num1)")

// Binary integer literal
let num2 = 0b101011
print("Binary Integer: \(num2)")

// Octal integer literal
let num3 = 0o52
print("Octal Integer: \(num3)")

// Hexadecimal integer literal
let num4 = 0x2C
print("Hexadecimal Integer: \(num4)")
```

Выход

```
Decimal Integer: 34
Binary Integer: 43
Octal Integer: 42
Hexadecimal Integer: 44
```

Swift-литералы с плавающей точкой

Литерал с плавающей точкой используется для представления числа с дробным компонентом, например, 34,567. Мы можем указать отрицательное или положительное значение в литерале с плавающей точкой, например, -23,45 или 2,34. Литерал с плавающей точкой может содержать подчеркивание (_), но это не оказывает никакого влияния на общее значение литерала, например, 0,2_3 и 23 оба являются одинаковыми. Литералы с плавающей точкой бывают следующих типов:

- **Десятичные литералы с плавающей точкой** — состоят из последовательности десятичных цифр, за которыми следует либо десятичная дробь, либо десятичная экспонента, либо и то, и другое. Например, 12.1875.
- **Шестнадцатеричные литералы с плавающей точкой** − состоят из префикса 0x, за которым следует необязательная шестнадцатеричная дробь, за которой следует шестнадцатеричный показатель степени. Например, 0xC.3p0.
- **Экспоненциальный литерал** — используется для представления степени числа 10. Содержит букву «e». Например, 1.243e4.

Пример

Программа на Swift для демонстрации литералов с плавающей точкой.

Открытый компилятор

```
import Foundation
```

```
// Decimal floating-point literal
let num1 = 32.14
print("Decimal Float: \(num1)")

// Exponential notation floating-point literal
let num2 = 2.5e3
print("Exponential Float: \(num2)")

// Hexadecimal floating-point literal
let num3 = 0x1p-2
print("Hexadecimal Float: \(num3)")
```

Выход

Decimal Float: 32.14
Exponential Float: 2500.0
Hexadecimal Float: 0.25

Строковые литералы Swift

Строковый литерал — это последовательность символов, заключенная в двойные кавычки, например, «символы». Строковые литералы могут быть представлены следующими способами −

- **Строка в двойных кавычках** − используется для представления однострочных литералов. Например, "привет".
- **Многострочная строка** − используется для представления многострочных литералов. Может содержать несколько строк без какого-либо экранирующего символа. Например −

```
"""Hello
I like your car """
```

Строковые литералы не могут содержать неэкранированную двойную кавычку ("), неэкранированную обратную косую черту (\), возврат каретки или перевод строки. Специальные символы могут быть включены в строковые литералы с помощью следующих escape-последовательностей &miinus;

Последовательность побега	**Имя**
\0	Нулевой символ
\\	\характер
\б	Возврат на одну позицию
\ф	Форма подачи
\н	Новая строка
\р	Возврат каретки
\т	Горизонтальная вкладка
\v	Вертикальная вкладка
\'	Одинарная кавычка
\"	Двойная кавычка
\000	Восьмеричное число от одной до трех цифр
\xxx...	Шестнадцатеричное число из одной или нескольких цифр

Пример

Программа на языке Swift для демонстрации строковых литералов.

```
Открытый компилятор
import Foundation

// Double-quoted string literal
let str1 = "Swift is good programming language"
print(str1)

// Multi-line string literal
let str2 = """
   Hello
   priya is at Swift programming
   """
print(str2)

// Special characters in a string literal
let str3 = "Escape characters: \n\t\"\'\\"
print(str3)
```

Выход

Hello
priya is at Swift programming
Escape characters:
"'\

Булевы литералы Swift

ТБулевы литералы используются для представления булевых значений true и false. Они обычно используются для представления условий. Булевы литералы бывают двух типов −

- **true** − представляет собой истинное состояние.
- **ложь** − представляет собой истинное состояние.

Пример

Программа на Swift для демонстрации булевых литералов.

```
Открытый компилятор
import Foundation

// Boolean literals in Swift
let value1 = true
let value2 = false

print("Boolean Literal: \(value1)")
print("Boolean Literal: \(value2)")
```

Выход

Boolean Literal: true Boolean Literal: false

Свифт – Комментарии

Комментарии — это специальный текст в программах, который не компилируется компилятором. Основная цель комментариев — объяснить нам, что происходит в конкретной строке кода или во всей программе. Программисты обычно добавляют комментарии, чтобы объяснить строку кода в программе.

Или мы можем сказать, что комментарии — это неисполняемый текст, и они похожи на заметку или напоминание для пользователя или программиста. В Swift мы можем определить комментарии тремя разными способами −

- Однострочные комментарии
- Многострочные комментарии
- Вложенные многострочные комментарии

Однострочный комментарий в Swift

Однострочный комментарий используется для добавления только однострочного текста в код. Однострочный комментарий начинается с двух косых черт (//). Компилятор или интерпретатор всегда игнорирует их и не влияет на выполнение программы.

Синтаксис

Ниже приведен синтаксис однострочного комментария:

```
// Add your comment here
```

Пример

Программа Swift для добавления однострочного комментария. Здесь мы добавляем однострочный комментарий в программу, чтобы объяснить работу цикла for-in.

Открытый компилятор

```
import Foundation
let num = 7
let endNum = 10

// For loop to display a sequence of numbers
for x in num...endNum{
   print(x)
}
```

Выход

7
8
9
10

Многострочный комментарий в Swift

Многострочные комментарии используются для отображения нескольких строк неисполняемого текста в программе, чтобы объяснить работу определенной строки кода или добавить предупреждения, примечания и т. д. разработчиками. Как и в других языках программирования, в Swift многострочные комментарии начинаются с косой черты, за которой следует звездочка (/*), и заканчиваются звездочкой, за которой следует косая черта (*/).

Синтаксис

Ниже приведен синтаксис многострочного комментария:

```
/* Add your
Mult-line comment here */
```

Пример

Программа Swift для добавления многострочных комментариев. Здесь мы добавляем несколько строк комментариев в программу, чтобы объяснить, как сложить два массива одинаковой длины.

Открытый компилятор

```
import Foundation

let arr1 = [1, 4, 6, 2]
let arr2 = [3, 5, 2, 4]

var result = [Int]()

/* Check the length of the array.
If they are equal then we add them using the + operator
and store the sum in the result array */
if arr1.count == arr2.count {
   for i in 0..<arr1.count {
      let sum = arr1[i] + arr2[i]
      result.append(sum)
   }
   print(result)
} else {
   print("Arrays must of same length.")
}
```

Выход
[4, 9, 8, 6]

Вложенный многострочный комментарий в Swift
Начиная с Swift 4, в многострочный комментарий также включена новая функция — вложенный многострочный комментарий. Теперь вы можете вкладывать или добавлять многострочный комментарий внутрь другого многострочного комментария. Он может легко комментировать многие блоки, даже если блок содержит многострочные комментарии.
Синтаксис
Ниже приведен синтаксис вложенных многострочных комментариев:

```
/* Add your multi-line comment.
/* Add your nested multi-line comment. */
End multi-line comment */
```

Пример
Программа Swift для добавления вложенных многострочных комментариев. Здесь мы добавляем вложенные многострочные комментарии в программу, чтобы добавить альтернативный код добавления двух массивов.

Открытый компилятор

```
import Foundation
let arr1 = [1, 4, 6, 2]
let arr2 = [3, 5, 2, 4]

var result = [Int]()
/* Checks the length of the array.
If they are equal then we add them using the + operator
and store the sum in the result array
/*You can also use the following code to add two arrays:
if arr1.count == arr2.count {
   let result = zip(arr1, arr2).map(+)
   print(result)
*/
*/
if arr1.count == arr2.count {
   for i in 0..<arr1.count {
      let sum = arr1[i] + arr2[i]
      result.append(sum)
   }
   print(result)
} else {
   print("Arrays must of same length.")
}
```

Выход
[4, 9, 8, 6]