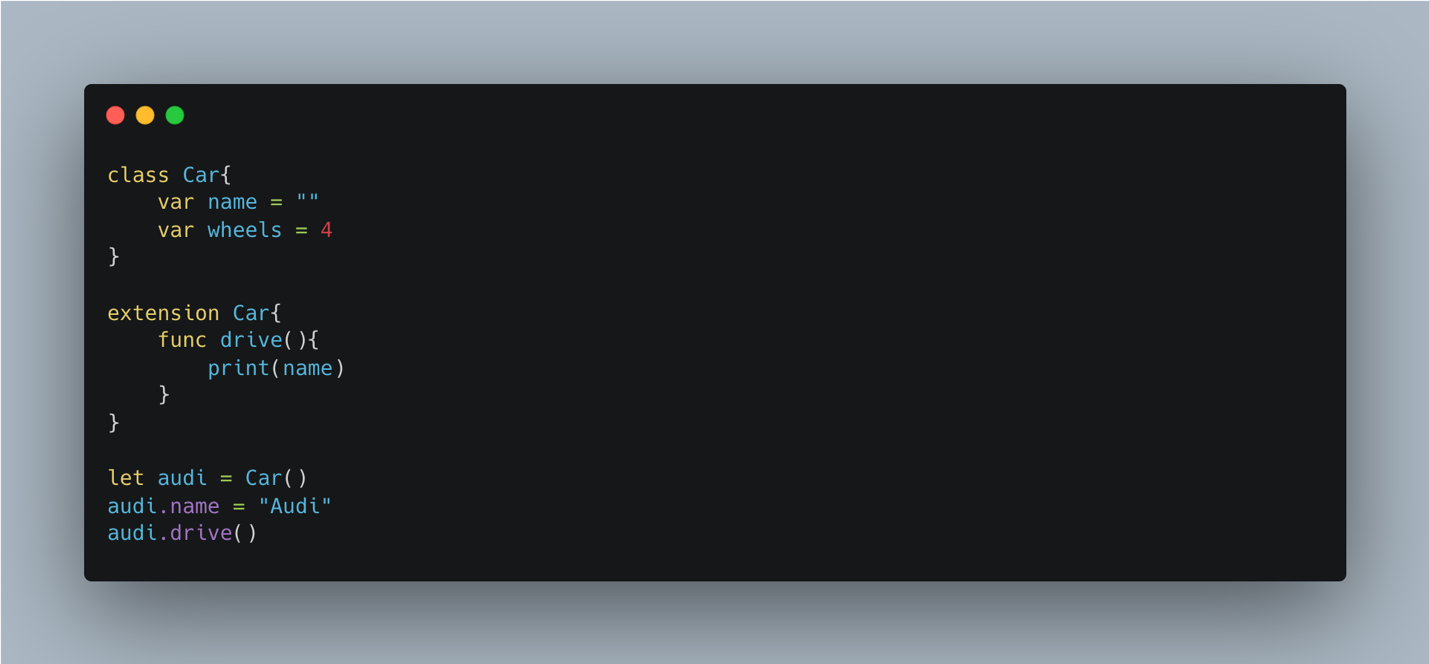
**Урок 7**

* Расширение типов
* Контроль доступа
* Замыкания

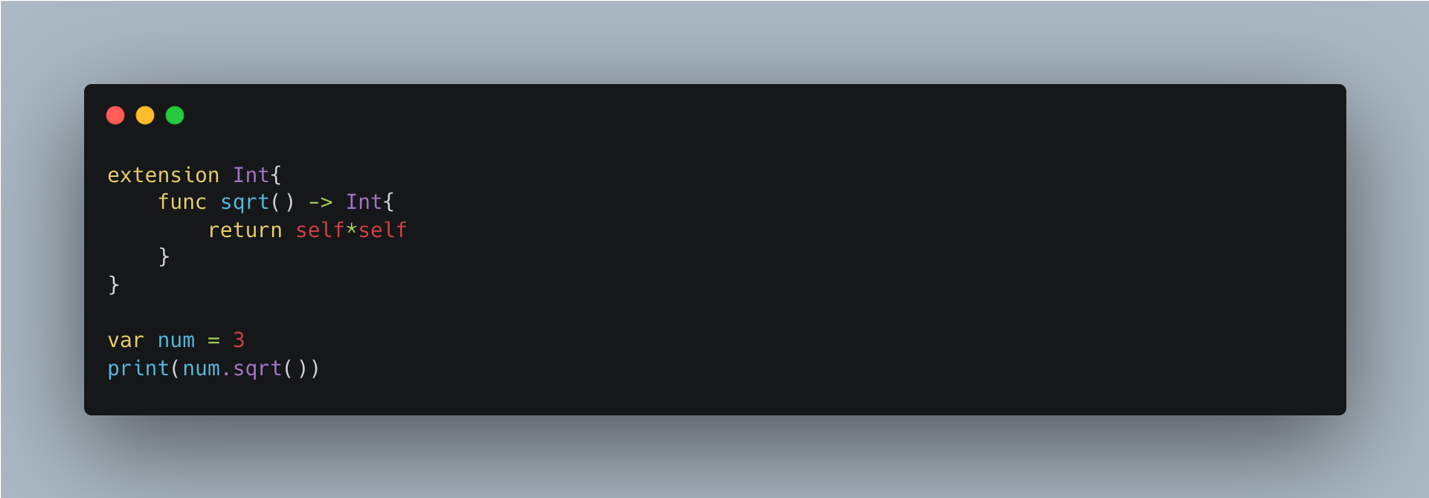
**Расширение типов**

Часто бывает, что в одном классе или структуре, у нас накапливается много кода, или бывает необходимо дописать функционал, но прямого доступа к объекту у нас нет. В таком случае существует возможность расширять тип данных. Для расширения в swift существует ключевое слово extension. Рассмотрим на примере:



В этом примере мы создали класс Car и задали ему свойства, а в расширении класса добавили еще и метод. Теперь у экземпляра класса есть доступ как свойствам так и к методам.

Еще одно применение, как было сказанно выще, это расширять типы данных, к которым у нас нет доступа. Один из принципов SOLID принцип открытости-закрытости гласит, что классы должны быть открыты для расширения, но закрыты для изменения. Как раз это мы можем реализовывать через расширения. К примеру, мы не можем изменить структуру Int, но можем ее расширить, добавить нужные нам методы, давайте расширим Int и добавим возможность выводить корень числа.



Через extension мы расширили тип данных Int, и теперь у него есть новый метод sqrt. Self внутри метода это передаваемое число, так как мы применили метод к числу 3 то соответственно под self будет как раз число 3.

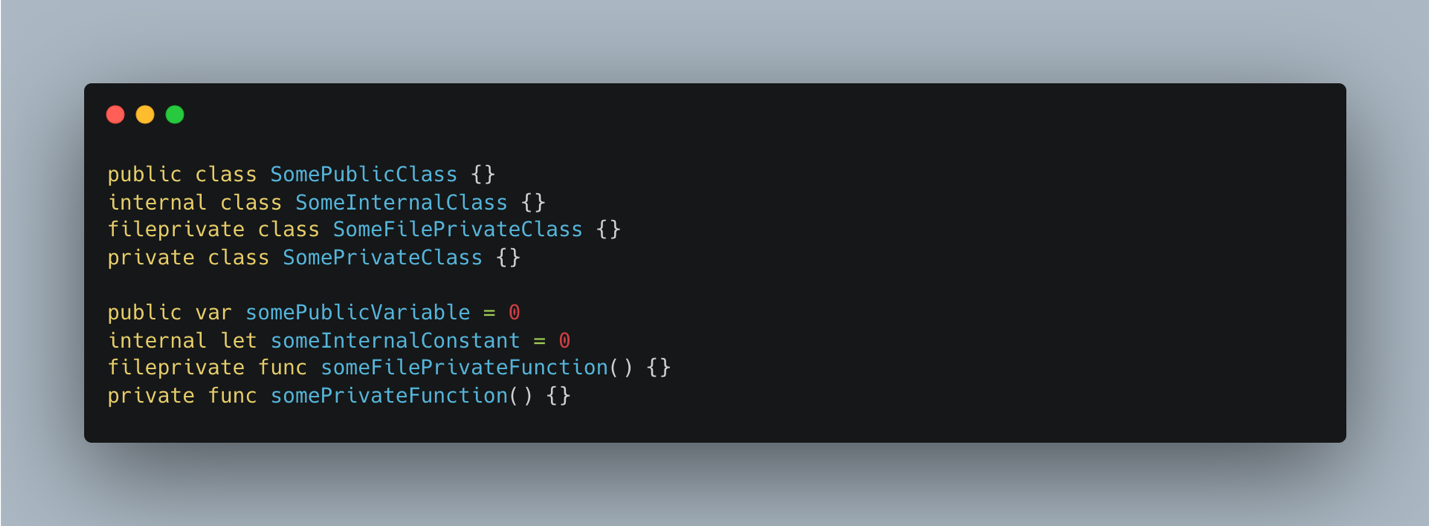
Расширения мы так же используем что бы выносить методы делегата за пределы класса, но это больше носит рекомендательный характер.

**Контроль доступа**

В ООП один из главных принципов – это инкапсуляция. Этот принцип гласит – скрывать все от пользователя все, что может изменяться в классе.

Допустим у нас в классе есть функция для отправки запроса на сервер, так же есть свойство, которое хранит в себе ссылку. Если мы сделаем экземпляр класса, то все методы и свойства сразу станут доступны, а допустим мы не хотим, чтобы к некоторым свойствам был доступ извне, для этого, как и в любом языке, в swift есть возможность ограничить зону видимости таких свойств, давайте разберем каждый:

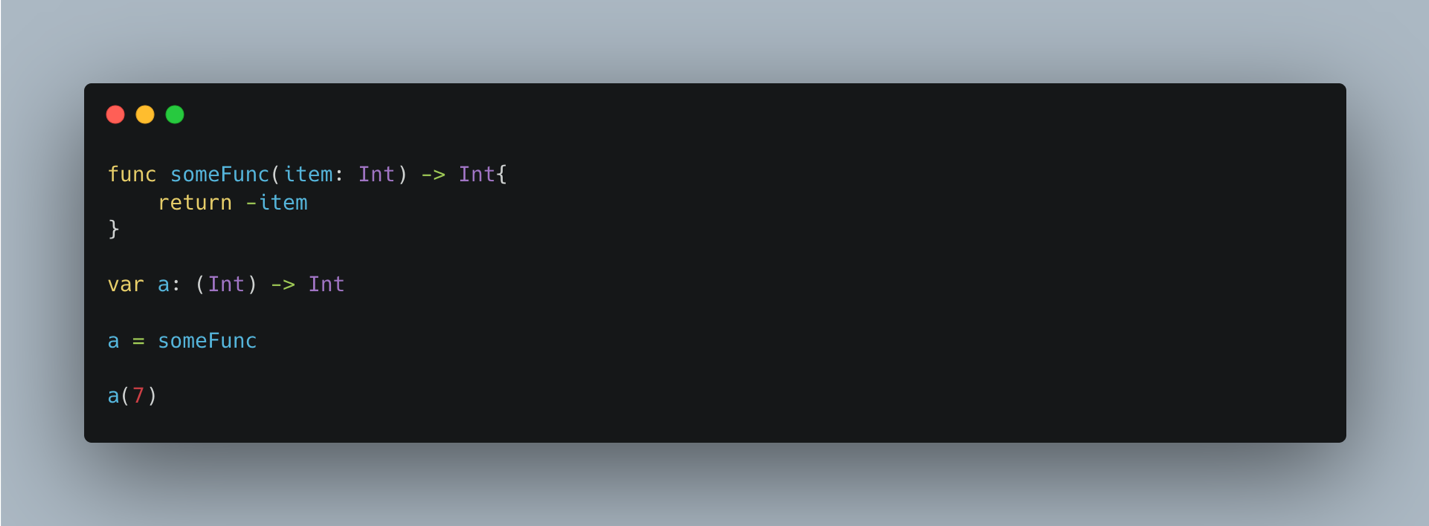
1. Open – используется при создании фреймворков, что бы можно было наследоваться от класса, на первых порах не будем использовать
2. Public – самый открытый тип объект можно использовать в любом классе, или в любом классе подключаемого модуля.
3. Internal – похож на public, отличие в том, что нельзя использовать объект в исходном файле не из этого модуля. По умолчанию.
4. Fileprivate – можно использовать объект только в пределах исходного файла
5. Private – можно использовать сущность только в пределах ее реализации, иначе говоря только в родительском классе.



Пример использования уровней доступа.

**Замыкания (closure)**

В Swift в качестве типа данных, мы можем присвоить так же тип функции, и передавать ее в другую функцию, это бывает полезно, когда часть функционала нужно вынести за пределы функции, замыкания работают схожим образом, только мы передаем анонимную функцию в качестве аргумента другой функции.



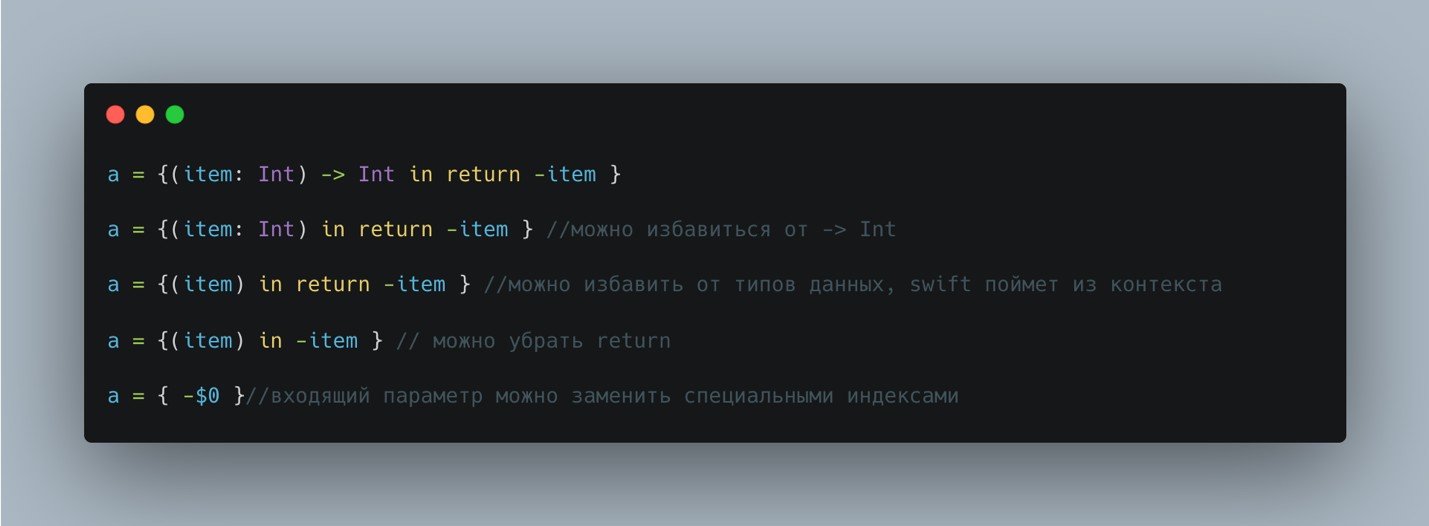
Мы создали простую функцию типа (Int) -> Int, а так же переменную такого типа, это значит, что мы можем присвоить к переменной функцию и вызывать ее через новую переменную.

Мы можем пойти дальше, и не объявлять функцию, а потом присваивать, а сразу определить ее в переменной:



Мы теперь точно так же можем вызвать функцию через переменную и результат будет таким же. Как и говорили выше, клоужер это безымянная функция, как на примере, который мы только что посмотрели. Теперь можно создать новую функцию, которая будет в качестве аргумента принимать некую безымянную функцию типа (Int) -> Int.

Синтаксис клоужеров в максимальном упрощении, давайте посмотрим, как можно упростить рассматриваемый пример:



Рассмотрим реализацию функции, которая принимает массив и клоужер и возвращает новый массив, все элементы которого четные.



ДЗ.

1. Напишите функцию, которая принимает словарь типа [String: Int] и возвращает словарь такого же вида. Так же функция принимает клоужер, через который необходимо реализовать проверку значений, больше или меньше определенного числа. (К примеру, вернуть словарь, где все значения больше 100 или вернуть словарь, где меньше 50. На каждое условие можно вызвать функцию по новой)

Ссылки:

<https://swiftbook.ru/content/languageguide/closures/>