Урок 5

Функции

Определение функции

> *Функция – это блок кода с заданным именем, к которому можно обратиться вызвав его по имени в нужный момент.*

Функции позволяют избежать дублирования кода за счет его группировки. Они помогают структурировать код, объединяя его повторяющиеся блоки для дальнейшего их использования.

Объявление функции

Синтаксис

```
func nameOfFunction() {
    some code
}
```

Объявление функции начинается с ключевого слова **func**, за которым идет имя функции. Функция может принимать параметры и возвращать некоторые значения. Параметры функции перечисляются в круглых скобках сразу после имени функции, а возвращаемые значения указываются после скобок и соответ-

ствующего символа. Но параметров и возвращаемых значений может и не быть. В примере выше приведен пример синтаксиса функции, которая не принимает ни каких параметров и не возвращает ни каких значений. В этом случае содержимое скобок остается пустым, а сразу после круглых скобок, между фигурными, помещается тело функции.

Для вызова функции используется её имя. Сразу после имени функции должны быть круглые скобки, даже если между ними ни чего нет. После каждого вызова функции исполняется код, помещенный в тело функции.

Все переменные и константы созданные внутри функци доступны только внутри этой функции. Мы не можем использовать их за пределами фигурных скобок вне функции. Фигурные скобки - это зона их видимости. Каждая переменная или константа доступна только в пределах этих скобок.

Параметры функций и возврат значений

Функции могут возвращать результат своей работы.

```
func nameOfFunction() -> Data Type {
    some code
    return some value
}
```

В отличии от предыдущего примера в этом после названия функции и круглых скобок ставится результирующая стрелка после которой указывается тип возвращаемого значения функции. В теле самой функции необходимо обязательно вернуть значение. Для этого последней строкой в блоке кода пишется ключевое слово **return** после которого значение для возврата. Тип значения должен соответствовать типу, который мы указываем после результирующей стрелки.

Функции могут принимать параметры:

```
func name(argumentOne parametrName: Data Type, argumentTwo
parametrName: Data Type) {
    some code
}
```

Список входных параметров заключается в круглые скобки и состоит из элементов, которые разделяются запятыми. Каждый отдельный элемент описывает отдельный входной параметр и состоит из имени и типа этого параметра, разделенных двоеточием. Входные параметры могут иметь различные типы и позволяют передать в функцию значения, которые ей требуются для реализации логики. Указанные параметры являются локальными, т.к. они доступны только внутри тела функции. Количество входных параметров может быть произвольным. Имя функции читается вместе с её параметрами. Параметры всегда имеют имя, но еще могут иметь и имена аргументов. Имя аргумента служит вспомогательным элементом т.е. оно несёт смысловую нагрузку на параметр. Имена параметров используются в теле функции, как константы. При вызове функции имена параметров заменяются значениями. Имена параметров так же записываются в стиле камэлкейс. **argumentOne** и **argumentTwo** это имена аргументов, которые останутся при вызове функции, тогда как **parametrName: Data Type** заменится значением того типа, который необходим.

Вариативные параметры

Вариативным называют параметр, который может иметь сразу несколько значений или не иметь ни одного. С помощью вариативного параметра можно передать в функцию произвольное число входных значений. Чтобы объявить параметр как вариативный, нужно поставить три точки (...) после его типа.

Значения, переданные через вариативный параметр, доступны внутри функции в виде массива соответствующего типа. Например, вариативный параметр numbers типа Double... доступен внутри функции в виде неизменяемого массива numbers типа `[Double]`.

У функции может быть только один вариативный параметр.

Вложенные функции

Существуют локальные функции, которые можно создать внутри других функций. Такие функции называются вложенными и обладают ограниченной областью видимости, т.е. доступны только внутри родительской функции.

Замыкания

В официальной документации Apple сказано, что замыкания или **closures** - это организованные блоки с определенным функционалом, которые могут быть переданы и использованы в вашем коде.

Замыкания бывают трех видов:

- Глобальные функции – это замыканиями, у которых есть имя и которые не захватывают никакие значения. Ни чего не напоминает? Да да это и есть те самые функции, которые мы изучили только что. То есть функции, являются частным случаем замыканий.
- Вложенные функции – это замыканиями, у которых тоже есть имя, но при этом они могут использовать или захватывать значения из родительской функции. И с этим видом замыканий мы уже знакомы.
- Замыкающие выражения - это безымянные функции, которые написаны в облегченном синтаксисе, которые могут захватывать значения из окружающего контекста

Конкретно под замыканиями понимают функции, которые не имеют имени, т.е. это сгруппированный в контейнер код, который может быть передан в виде аргумента и многократно использован.

Синтаксис замыкающего выражения

```
{ (параметры) -> тип результата in
    тело замыкающего выражения
}
```

Замыкающее выражение пишется между фигурными скобками. После входных параметров ставится результирующая стрелка и тип возвращаемого значения, а затем ключевое слово **in**. После ключевого слова следует тело замыкания, по аналогии с телом функции.

Захват значений замыканиями

Замыкания могут захватывать значения констант и переменных из окружающего контекста, в котором оно объявлено. После захвата замыкание может ссылаться на эти значения или модифицировать их внутри своего тела, даже если область, в которой были объявлены эти константы и переменные уже больше не существует.