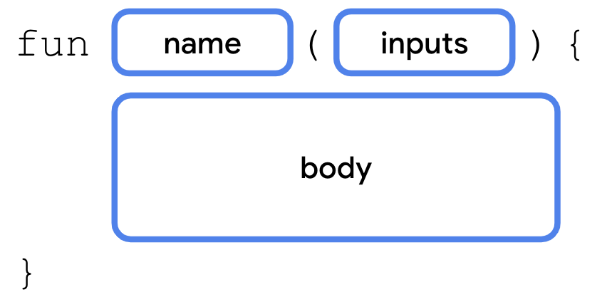
Базовый синтаксис Kotlin

# Основные положения

Выполнение программы начинается с определенного места – точки входа программы (главной функции). Такой функцией в Kotlin является main(). Как и в Java, главной функции можно передать массив пустых строк в качестве параметра, однако, в данном языке программирования это не является обязательным.

Для того, чтобы определить блок кода как функцию, необходимо использовать ключевое слово fun.

Общий вид функции представлен на рисунке ниже:



Функция println() является системным методом для вывода информации в консоль.

fun main() {

println("Hello world!")

}

Для пользовательского ввода используется метод системной библиотеки readln().

println("Enter any word: ")

// Reads and stores the user input. For example: Happiness

val yourWord = readln()

// Prints a message with the input

print("You entered the word: ")

print(yourWord)

# ЗАДАНИЯ

В качестве практики работы с ошибками с помощью компилятора, скопируйте и вставьте следующий код:

**fun main() {  
    println("Today is sunny!)  
}**

Внимательно прочитайте то, что выводится в консоли. Компилятор указывает, как файл, в котором замечена ошибка, так и номер строки и позицию, где произошла ошибка (номер строки и символа)

# Переменное количество параметров. Vararg

Функция может принимать не только ограниченное количество параметров определенного типа, например:

printStrings(vararg strings: String)

# ЗАДАНИЯ

Прочитайте код и попробуйте ответить на вопрос: каким будет результат выполнения следующего кода?

**fun printStrings(vararg strings: String){**

**for(str in strings)**

**println(str)**

**}**

**fun main() {**

**printStrings("Tom", "Bob", "Sam")**

**printStrings("Kotlin", "JavaScript", "Java", "C#", "C++")**

**}**

# Переменные

Переменные – повторяющиеся или используемые данные в приложении. Для первоначального понимания переменных, их представляют контейнерами, в которое помещают определенное значение.

Типы данных

В изучаемом языке программирования необходимо типизировать значения, для того, чтобы производить над ними специфические для типа операции.

Ниже приведены основные типы данных:

|  |  |  |
| --- | --- | --- |
| **Kotlin data type** | **What kind of data it can contain** | **Example literal values** |
| String | Text | "Add contact" "Search" "Sign in" |
| Int | Integer number | 32 1293490 -59281 |
| Double | Decimal number | 2.0 501.0292 -31723.99999 |
| Float | Decimal number (literal F or f at the end!) | 5.0f -1630.209f 1.2940278F |
| Boolean | true or false | true false |
| Char | Character | ‘A’  ‘x’ |

ВАЖНО:   
Типы – классы. Пишутся с большой буквы.

Определение и использование

Когда в программе подразумевается использование конкретной переменной, говорится о ее «определении» (объявлении). Рисунок ………..



fun main() {  
    val count: Int = 2  
    println(count)  
}

В Kotlin, для определения переменной (как и в случае с функциями) используются ключевые слова val и var, которые обозначают возможность изменения переменной по ходу выполнения программы

**val** – (иммутабельная) изменяемая переменная. Можно присвоить новое значение

**var** – (мутабельная) изменяемая переменная. Значение не меняется.

Приведение типов

Если переменная объявляется с присвоением значения, тип допускается не прописывать. То есть, следующий код отработает корректно:

val a: Long = 5

val b = a

Тип второй переменной присвоится автоматически – неявное приведение (преобразование) типов.

Явное приведение типов – указание компилятору вручную, какого типа переменная.

ВАЖНО:   
Существуют операции приведения одного типа к другому, однако, когда речь идет о приведении «большего типа» к «меньшему (например, из типа Double в Int) может произойти потеря данных

*Базовые операции для простых типов*

- - вычитание

+ - сложение

\* - умножение

/ - деление

% - остаток от деления

# Задание

Скопируйте и запустите код, иллюстрирующий частный случай деления целых чисел. Ответьте на вопрос: какого типа результат деления?

fun main() {  
 val a = 5  
 val b = 2  
 println((a / b)::class)  
}

*Операции с логическими значениями типа Boolean*

|| - логическое ИЛИ. Вернет true, если ХОТЯ БЫ ОДНО значение соответствует условию

&& - логическое И. Вернет true, если ОБА условия выполнятся

! – операция отрицания. Вернет преобразованное значение (!true -> false, !false -> true)

*Символы и строки*

Компилятор отличает строку от символа не только по длине, но так же по кавычкам. Для строк – “, для символов – ‘.

Строковые шаблоны

Строковые переменные предполагают работу с текстом, который, в основном, сохраняется в базе данных или выводится пользователю в качестве информации.

Например:

fun main() {  
    val count: Int = 2  
    println("You have unread messages: ")  
    println(count)  
}

Чтобы вывести сообщение в одну строчку, можно использовать строковый шаблон – строка, в которой используется переменная программы. Перед ней проставляется знак $, таким образом, компилятор считывает символы за этим знаком, находит соответствующую переменную и проставляет ее значение.

Если перед выводом необходимо провести некоторое вычисление с переменными, можно воспользоваться следующей записью:

fun main() {  
    val readMessages: Int = 2

    val AllMessagescount: Int = 10  
    println("You have unread messages: ${allMessages - readMessages}")  
}

*Операции преобразования чисел*

**Инкремент –** увеличение числа на единицу - ++.

**Декремент –** уменьшение числа на единицу – --.

Данные операторы могут стоять как до переменной, так и после переменной.

# Задание

Скопируйте и запустите код, иллюстрирующий использование инкремента и декремента в префиксном и постфиксном вариантах (соответственно). Ответьте на вопрос – чем отличаются префиксные и постфиксные операнды

**Var a = 5**

**Val b = a++**

**println(“a: $a, b: $b”)**

**Var c = 5**

**Val d = --a**

**println(“a: $a, b: $b”)**

*Математические операции и присвоение значений*

A **+=** B – увеличение переменной А на переменную В и присваивание нового результата

A **-=** B – уменьшение переменной А на переменную В и присваивание нового результата

A **\*=** B – умножение переменной А на переменную В и присваивание нового результата

A **/=** B – деление переменной А на переменную В и присваивание нового результата

A **%=** B – ltktybt переменной А на переменную В и присваивание остатка в качестве нового результата

*Конкатенация строк*

# Задание

Скопируйте код, вставьте в блок главной функции и запустите. Строка вывода иллюстрирует использование конкатенации. Ответьте на вопрос – каков результат конкатенации строк?

**Var a = “Hello,”**

**Val b = “World”**

**println(a + “,“ + “ ${b}”)**

# Комментирование кода

В процессе работы над программой существует необходимость текстового документирования блоков кода для фиксации специфики или других нюансов его выполнения.

Существует для вида комментариев кода:

Однострочный – представляет собой одну строку

// This is a comment.

Многострочный – комментарий, занимающий несколько строк.

/\*  
 \* This is a very long comment that can  
 \* take up multiple lines.  
 \*/

# ЗАДАНИЯ

Объявите иммутабельную переменную dogName, присвойте ей любую кличку животного, (например, “Rex”).   
Объявите иммутабельную переменную dogAge, присвойте число, соответствующее возрасту животного.  
Выведите в консоль сообщения:

**“Happy birthday, Rex”**

**“You are now 5 years old!”**

Не забудьте использовать строковые шаблоны

# Функции

Определение функции – описание ее имени, параметров и тела.

Вызов функции – обращение к ней по ее имени (с прописыванием круглых скобок).

Модифицируем функцию для вывода поздравления питомца:

fun main() {  
    birthdayGreeting()  
}  
  
fun birthdayGreeting() {  
    println("Happy Birthday, Rex!")  
    println("You are now 5 years old!")  
}

В крупных приложениях функции созданы не только для вывода текста.

Некоторые функции могут вернуть преобразованное значение для сохранения результата в новую переменную для дальнейшего использования его в коде.

Для того, чтобы функция вернула значение, необходимо прописать тип возвращаемых данных. После круглых скобокставится двоеточие и прописывается тип (по аналогии с переменными, после двоеточия и до оператора присваивания значения- = ). А так же в конце тела функции прописать оператор return и возвращаемое значение или выражение.

fun birthdayGreeting(): String {  
    val nameGreeting = "Happy Birthday, Rover!"  
    val ageGreeting = "You are now 5 years old!"  
    return "$nameGreeting\n$ageGreeting"  
}

Обратите внимание, что модифицированная функция фозвращает значение – это означает, что ее результат необходимо или записать в новую переменную:

fun main() {  
    val greeting = birthdayGreeting()  
    println(greeting)

Или вернуть как входное значение другой функции

fun main() {  
    println(birthdayGreeting())  
}

*Параметры функции*

Для функции println() передаваемым параметром для вывода служит строка, которую вы указывали внутри круглых скобок.

Параметры – переменные, используемые в теле функции. Сначала необходимо их объявить в круглых скобках после имени функции. Указываются имя и тип параметра (после двоеточия).

fun birthdayGreeting(name: String): String {  
    val nameGreeting = "Happy Birthday, $name!"  
    val ageGreeting = "You are now 5 years old!"  
    return "$nameGreeting\n$ageGreeting"  
}

Если функция принимает несколько аргументов, они указываются последовательно, через запятую.

# ЗАДАНИЯ

Модифицируйте функцию для поздравления.

Добавьте в параметры функции переменную **age** числового типа.

Используйте переменную **age** в сообщении, где проставляется возраст питомца.

Не забудьте использовать строковые шаблоны.

Скопируйте код программы:

**fun main() {  
 functionWithError(false, "NewFunction")  
}  
  
fun functionWithError(name: String, hasError: Int): String {  
 return "My function - $name has error - $hasError!"  
}**

Исправьте ошибку компиляции, чтобы функция вывела сообщение:

**My function - NewFunction has error - false!**

Внимательно читайте сообщения компилятора в консоли!

# Сигнатура функции

Имя функции и ее параметры (до определения типа функции) называют ее сигнатурой.

# Именованные аргументы

При вызове функции важно указывать параметры в соответствии с их порядком объявления в сигнатуре. Однако, если по какой-то причине, разработчик не может явно посмотреть порядок вызова параметров, можно воспользоваться «именованием аргументов». То есть, прописать имена передаваемым значениям, в соответствии с именем аргументов:

functionWithError(**hasError =** false, **name =** "NewFunction")

ЗАДАНИЯ: Базовый синтаксис Kotlin

* Исправьте ошибки компиляции:

fun main() {   
    println("New chat message from a friend'}  
}

* Исправьте ошибки компиляции:

fun main() {   
    println(‘New chat message from a hater’}  
}

* Определите иммутабельную числовую переменную, обозначающую процент чаевых, с именем percentTip, присвойте ей значение 15.  
  Определите иммутабельную числовую переменную, обозначающую цену на кофе, с именем coffeePrice, присвойте значение 180.  
  Определите иммутабельную числовую переменную, обозначающую итоговую стоимость пяти выпитых чашек кофе, с именем totalSum.  
  Выведите в консоли сообщение:

Your totalSum: 900. Tip count for officiant: 135

* Вынесите дублирующийся код в функцию:

fun main() {  
    println("City: Ankara")  
    println("Low temperature: 27, High temperature: 31")  
    println("Chance of rain: 82%")  
    println()  
  
    println("City: Tokyo")  
    println("Low temperature: 32, High temperature: 36")  
    println("Chance of rain: 10%")  
    println()  
      
    println("City: Cape Town")  
    println("Low temperature: 59, High temperature: 64")  
    println("Chance of rain: 2%")  
    println()  
      
    println("City: Guatemala City")  
    println("Low temperature: 50, High temperature: 55")  
    println("Chance of rain: 7%")  
    println()  
}

* Скопируйте код

fun main() {  
 val name = "Rex"  
 val foodType = 'vegetable'  
  
 println("$name doesn't like $foodTypes")  
}

Исправьте ошибки, чтобы функция выводила сообщение: **Rex doesn't like vegetables**

Управляемый поток исполнения

# Операторы сравнения

*Сравнение значений*

Для обозначения сравнения пременных а и b используются следующие операторы:

**== -** вернет true, если значения равны

**!= -** вернет true, если значения не равны

**>, < -** вернет true, если значение слева больше или меньше, чем значение справа (соответственно)

**>=, <= -** вернет true, если значение слева больше (или равно) или меньше (или равно) , чем значение справа (соответственно)

*Проверка типа*

# ЗАДАНИЯ

Скопируйте код и вставьте в тело главной функции.

Ответьте на вопрос: какой оператор выполняет проверку типа переменной?

**var a = "Hello"  
println(a is String)**

# Ветвление

Если необходимо выполнить разные операции в соответствии с результатом вычисления, Kotlin предлагает такие операторы сравнения, как if/else и when/else.

*If – else*

Операторы ветвления призваны для того, чтобы выполнять определенные операции при соблюдении условия блока if или else.

Конструкция выглядит следующим образом:

if (…) {

// code

} else {

// code

}

Конструкция может возвращать значение. Таким образом, можно присваивать результат непосредственно в переменную или использовать конструкцию, как входной параметр функции.

Допускается использовать только блок if, else в данном случае не прописывается.  
Пример реализации ветвления:

fun main() {  
 val a = 1  
 val b = 2  
  
 // without else  
 if (a < b) println("test: a less than b")  
  
 val maxVariable: String;

// has else  
 if (a > b) {  
 maxVariable = "a"  
 } else {  
 maxVariable = "b"  
 }  
 println("Max variable is $maxVariable")  
  
 // inline  
 val max = if (a > b) a else b  
 println("max equal is $max")  
}

Для возврата значения в переменную, достаточно прописать возвращаемое значение в конце блока:

val maxVariable = if (a > b) {  
 /\*  
 \* other code  
 \* \*/  
 a   
} else {  
 /\*  
 \* other code  
 \* \*/  
 b  
}

*When*

Когда необходимо сравнить значение по нескольким параметрам, применяют оператор when.

fun main() {  
 val x = -1  
  
 val res: String;  
  
 when (x) {  
 0 -> res = "zero"  
 1 -> res = "positive and not zero"  
 else -> {  
 res = "negative"  
 }  
 }  
  
 println("$res")  
}

Оператор сравнивает значение в скобках с теми аргументами, что стоят слева (до оператора ->), в случае, если ни одно из условий не пройдено, выполнится блок else. Обратите внимание, что блок альтернативного условия находится внутри when.

Если определенные действия должны выполняться по нескольким значениям, допускается их перечисление через запятую:

1, 2, 3 -> res = "positive and not zero"

Как и if, when так же возвращает значения.

# ЗАДАНИЯ

Модифицируйте главную функцию – присвойте результат блока when сразу при объявлении переменной res.

**fun main() {  
 val x = 0  
  
 val res: String;  
  
 when (x) {  
 0 -> res = "zero"  
 1 -> res = "positive and not zero"  
 else -> {  
 res = "negative"  
 }  
 }  
  
 println("$res")  
}**

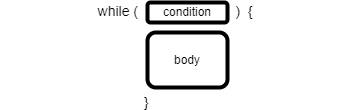
# Циклы

Циклы – выполняемые блоки кода столько раз, сколько выполняется прописанное условие.

ВАЖНО: Применение циклов держит код чистым и читаемым. Однако, стоит аккуратно прописывать условия исполнения, чтобы программа дошла до выхода из цикла. Иначе исполнение завершится с ошибкой при компиляции.

While

Конструкция выглядит следующим образом:



# ЗАДАНИЯ

Попробуйте ответить на вопросы: каким будет результат выполнения следующего кода?

Что произойдет, если убрать постфиксный инкремент?

**var i = 0**

**while (i < 5) {**

**println(i)**

**i++**

**}**

*Do | while*

Одна из разновидностей циклов. В данном случае блок кода выполняется до проверки условия, затем выполняется столько раз, сколько условие будет выполняться.

# ЗАДАНИЯ

Попробуйте ответить на вопрос: каким будет результат выполнения следующего кода?

**var i = 0**

**do {**

**println(i)**

**i++**

**}**

**while (i < 5)**

*break | continue*

Существуют операторы, позволяющие прервать стандартное выполнение цикла:

* break – позволяет прервать выполнение цикла и выйти из него
* continue – позволяет «перепрыгнуть» определенные шаги цикла

# ЗАДАНИЯ

Попробуйте ответить на вопрос: каким будет результат выполнения следующего кода?

**var i = 0**

**while (i < 10) {**

**if (i == 4) {**

**i++**

**continue**

**}**

**println(i)**

**i++  
}**

Коллекции

В рамках примера с поздравлением питомцев, представим, что есть несколько животных, которых поочередно нужно поздравить. Чтобы не создавать n-количество переменных для сохранения кличек и возраста, можно обратиться к коллекциям – структура данных, систематизированная подборка данных, собранных по определенному признаку.

# Массивы

Базовая коллекция. Содержит n-количество данных. Представлен классом Array.

var dogsArray: Array<String> = arrayOf("Nile", "Rex", "Rover")

// empty Array

var exampleArray = emptyArray<String>()

Для доступа к определенному элементу используются индексы – числовой показатель места в массиве. Их прописывают в квадратных скобках:

var dogsArray: Array<String> = arrayOf("Nile", "Rex", "Rover")

println(dogsArray[2])

*Доступ и модификация*

Простые операции добавления нового элемента в конец структуры и модификация элемента по индексу:

fun main() {  
 var dogsArray: Array<String> = arrayOf("Nile", "Rex", "Rover")  
  
 dogsArray **+=** "Luke"  
 // Nile, Rex, Rover, Luke  
 println(dogsArray.joinToString())  
  
 dogsArray[0] = "Fluffy"  
 // Fluffy, Rex, Rover, Luke  
 println(dogsArray.joinToString())  
}

Для того, чтобы добавить массив значений в новый массив, можно воспользоваться оператором «спред» - \*. Этот оператор поочередно вставляет элементы массива как индивидуальные аргументы

fun main() {

val lettersArray = arrayOf("c", "d")

val newArray = arrayOf("a", "b", **\*lettersArray**) // abcd

}

*Вложенные массивы*

Существуют массивы с разным уровнем вложенности данных – когда одному элементу верхнего массива соответствует другой массив. Например, двумерные массивы:

// [[a, b], [c, d], [e, f]]  
fun main() {  
 var literalArray: Array<Array<String>> = arrayOf(  
 arrayOf("a", "b"),  
 arrayOf("c", "d"),  
 arrayOf("e", "f")  
 )  
 println(literalArray[2][0])  
}

Вложенные массивы могут иметь многие уровни вложенности, такие массивы называют многомерными

*Использование*

Массивы считаются примитивной структурой. Об использовании массивов говорят, когда структура представляемых данных «плоская» (не предполагает уровней вложенности), однако и в этих случаях разработчики отдают предпочтения другим типам коллекций, поскольку те предоставляют более широкий функционал взаимодействия и модификации.

*Функции класса*

Поскольку массивы – элементы класса, им доступны некоторые функции, например, сравнение, конвертирование в другие типы и т.д.

**Проверить наличие элемента в массиве**

Оператор in проверяет вхождение элемента в массив:

val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")

if ("Volvo" in cars) {

println("It exists!")

} else {

println("It does not exist.")

}

# ЗАДАНИЯ

Вспомните теорию логических операторов и ответьте на вопрос: как модифицировать условие в выражение «элемент ’Volvo’ не содержится в массиве».

**Пройти по каждому элементу**

Часто в работе есть необходимость совершить определенные действия для каждого элемента массива. Применяется с использованием оператора for.

val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")

for (x in cars) {

println(x)

}

ВАЖНО: обратите внимание, в языке Kotlin данный тип цикла выглядит иначе в отличие, например, от Java.

# Диапазоны (ranges)

Диапазон – набор значений, интервал.

Диапазон создается при помощи оператора ...

val range = 1..5    // диапазон [1, 2, 3, 4, 5]

Оператор принимает две границы – нижнюю и верхнюю (нижняя – 1, верхняя - 5).

Описанный оператор создает элементы, которые больше предыдущего (это же касается создания диапазона строковых символов).

Для того, чтобы создать обратный диапазон – где каждый последующий элемент меньше предыдущего, воспользуйтесь оператором downTo.

val range1 =  1..5      // 1 2 3 4 5

val range2 =  5 downTo 1    // 5 4 3 2 1

Если необходимо создавать значения через определенный шаг, используют оператор step.

# ЗАДАНИЯ

Попробуйте ответить на вопрос: каким будет результат выполнения следующего кода?

**val range1 = 1..10 step 2**

Если необходимо исключить верхнюю границу в диапазон, используют оператор until.

# ЗАДАНИЯ

Попробуйте ответить на вопрос: каким будет результат выполнения следующего кода?

**val range2 = 1 until 9 step 2**

# List, Set, Map

**List –** набор элементов. Доступ осуществляется по индексу (числовой показатель позиции элемента). Элементы в списках могут повторяться.

**Set –** коллекция уникальных элементов (повторения отсутствуют).

**Map –** набор пар ключ – значение. Ключи – уникальные показатели, по которым осуществляется доступ к элементам. Значения могут повторяться. «Мапы» используются для сохранения логического доступа между объектами. Например, ID сотрудника и его должности.

Kotlin предоставляет возможность работы с коллекциями независимо от типов, которые они сохраняют. Другими словами – в коллекцию элементов String вы можете добавить как элементы этого типа, так и других (например, Int или значение пользовательского типа)

Коллекции реализуют два интерфейса:

* read-only – неизменяемые коллекции.

Данный интерфейс позволяет в цикле перебрать или проверить элементы.

Методы:   
size – размер коллекции;   
isEmpty() – вернет true, если коллеция не содержит элементы;   
contains(elem) – вернет true, если коллекция содержит элемент;   
containsAll(collection) – вернет true, если коллекция содержит элементы коллекции collection.

* mutable – изменяемые коллекции.

Предоставляет работу по добавлению и удалению элементов.

Методы:

add(elem) – добавляет элемент;

remove(elem) – удаляет элемент;

addAll(elements) – добавляет набор элементов;

removeAll(elements) – удаляет набор элементов;

clear() – удаляет все элементы коллекции.

Подробнее тема коллекций рассматривается в пособии.

**https://kotlinlang.org/docs/kotlin-tour-classes.html#practice**

[**https://www.w3schools.com/KOTLIN/kotlin\_oop.php**](https://www.w3schools.com/KOTLIN/kotlin_oop.php)

[**https://kotlinlang.org/docs/kotlin-tour-null-safety.html**](https://kotlinlang.org/docs/kotlin-tour-null-safety.html)

**Дальше переходи к туториалу на андроид**