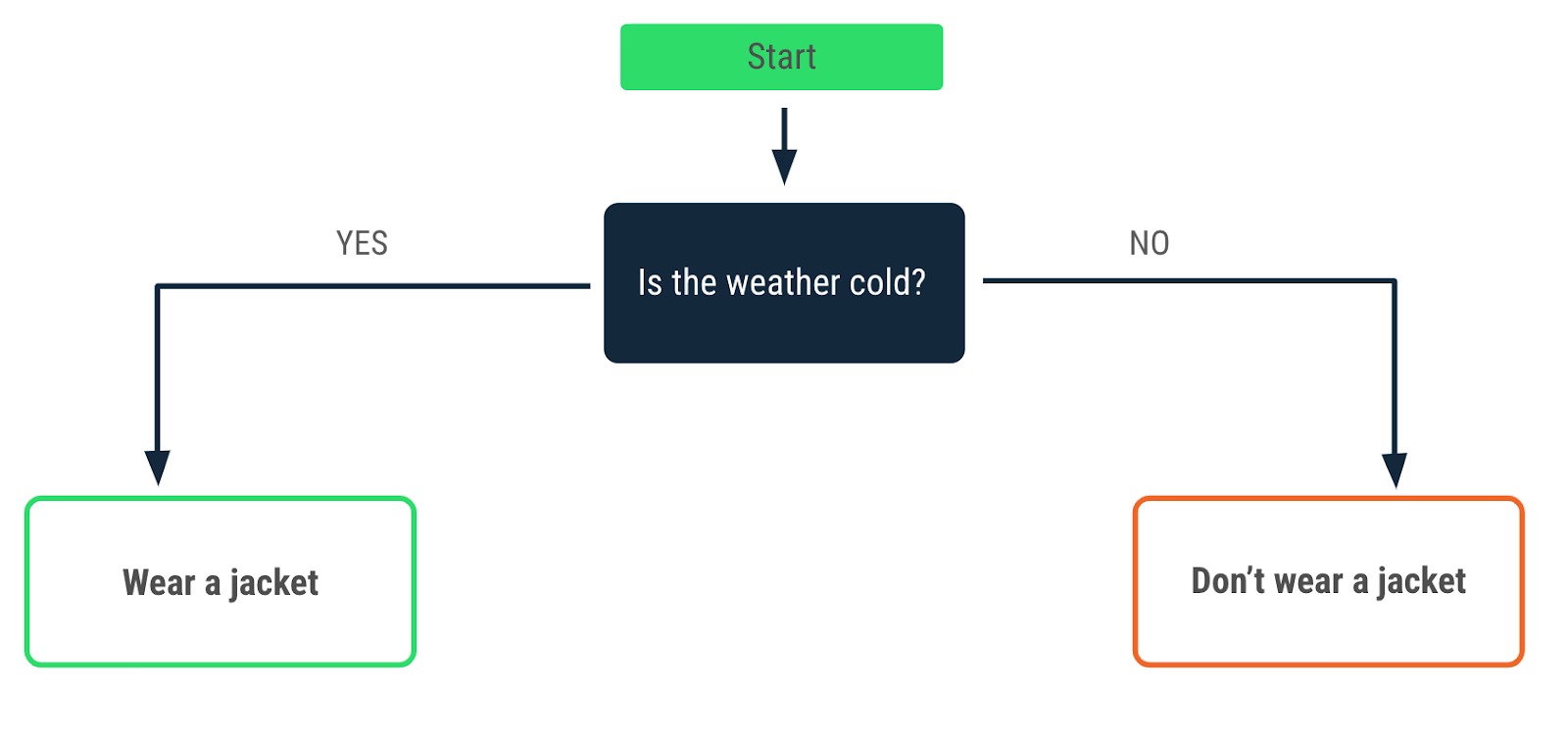
# **Основы Котлин**

6. <https://developer.android.com/courses/pathways/android-basics-compose-unit-2-pathway-1>

[2. Используйте операторы if/else для выражения условий.](https://developer.android.com/codelabs/basic-android-kotlin-compose-conditionals?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-conditionals#1)

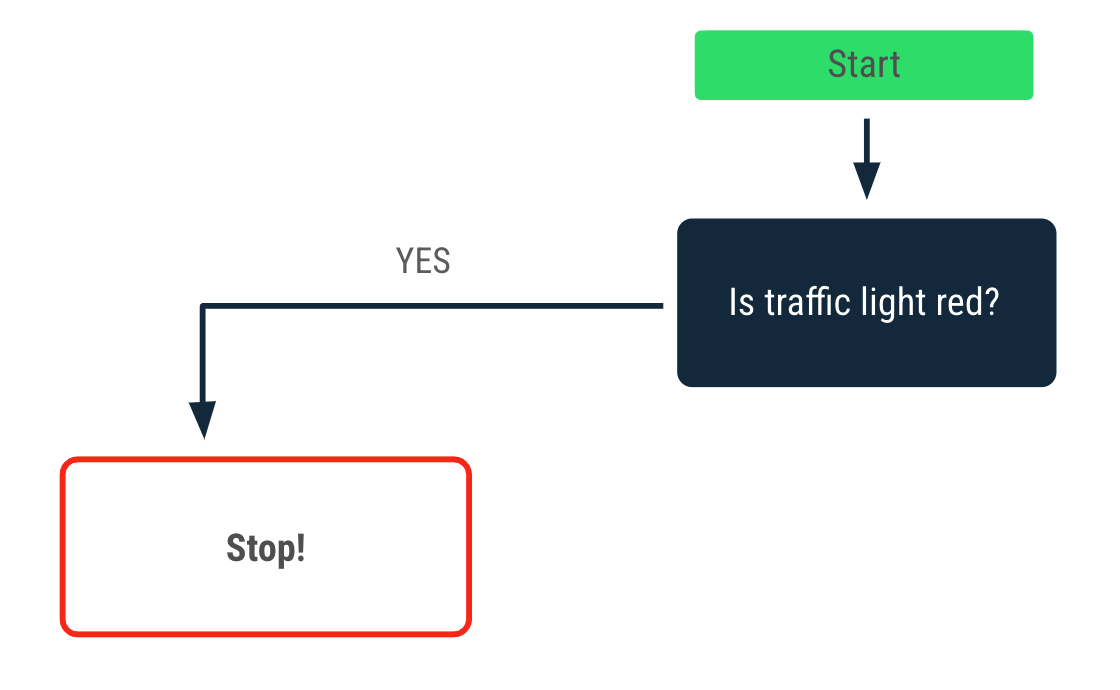
В жизни принято поступать по-разному в зависимости от ситуации, с которой вы сталкиваетесь. Например, если погода холодная, вы носите куртку, а если погода теплая, вы не носите куртку.



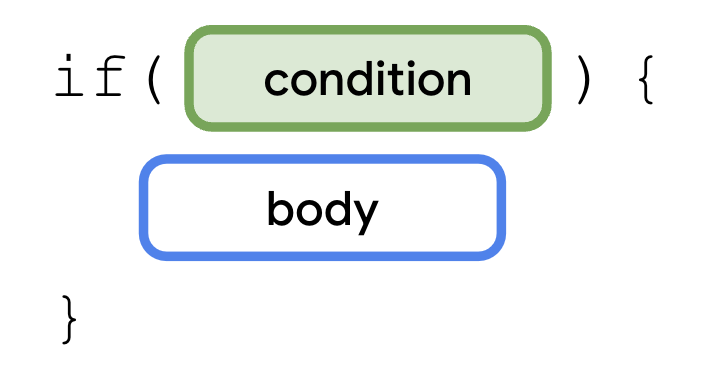
Принятие решений также является фундаментальной концепцией программирования. Вы пишете инструкции о том, как программа должна вести себя в конкретной ситуации, чтобы она могла действовать или реагировать соответствующим образом при возникновении ситуации. В Kotlin, если вы хотите, чтобы ваша программа выполняла различные действия в зависимости от условия, вы можете использовать if/elseоператор. В следующем разделе вы пишете ifзаявление.

Запись ifусловий с логическими выражениями

Представьте, что вы создаете программу, которая сообщает водителям, что им следует делать, когда они стоят на светофоре. Ориентируйтесь на первое условие: красный свет светофора. Что вы делаете на красном светофоре? Останавливаться!



В Котлине это условие можно выразить с помощью ifоператора. Взгляните на анатомию высказывания if:



Чтобы использовать ifоператоры, вам необходимо использовать ifключевое слово, за которым следует условие, которое вы хотите оценить. Вам нужно выразить условие с помощью логического выражения. *Выражения* объединяют значения, переменные и операторы, возвращающие значение. Логические выражения возвращают логическое значение.

Ранее вы узнали об *операторах присваивания* , таких как:

val number = 1

Оператор =присваивания присваивает numberпеременной 1значение.

Напротив, логические выражения создаются с помощью *операторов сравнения* , которые сравнивают значения или переменные в обеих частях уравнения. Взгляните на оператор сравнения.

1 == 1

Оператор ==сравнения сравнивает значения друг с другом. Как вы думаете, какое логическое значение возвращает это выражение?

Найдите логическое значение этого выражения:

1. Используйте [Kotlin Playground](https://developer.android.com/training/kotlinplayground" \t "_blank) для запуска вашего кода.
2. В тело функции добавьте функцию println()и затем передайте ей 1 == 1выражение в качестве аргумента:

fun main() {  
    println(1 == 1)  
}

1. Запустите программу и просмотрите результат:

true

Первое 1значение равно второму 1значению, поэтому логическое выражение возвращает значение true, которое является логическим значением.

Попробуй

Помимо ==оператора сравнения, существуют дополнительные *операторы сравнения* , которые можно использовать для создания логических выражений:

* Меньше, чем:<
* Больше чем:>
* Меньше или равно:<=
* Больше или равно:>=
* Не равно:!=

Попрактикуйтесь в использовании операторов сравнения с простыми выражениями:

1. В аргументе замените ==оператор сравнения на <оператор сравнения:

fun main() {  
    println(1 < 1)  
}

1. Запустите программу и просмотрите результат:

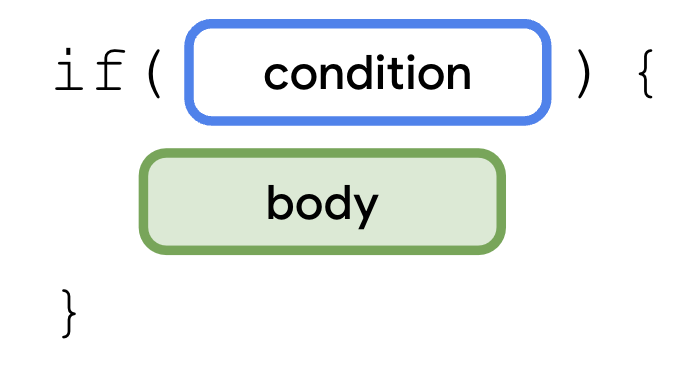
Выходные данные возвращают falseзначение, поскольку первое 1значение *не* меньше второго 1значения.

false

1. Повторите первые два шага с другими операторами сравнения и числами.

Напишите простое ifзаявление

Теперь, когда вы увидели несколько примеров написания логических выражений, вы можете написать свой первый ifоператор. Синтаксис оператора ifследующий:



Оператор ifначинается с ifключевого слова, за которым следует условие, которое представляет собой логическое выражение внутри круглых скобок и набора фигурных скобок. Тело — это серия операторов или выражений, которые вы помещаете в фигурные скобки после условия. Эти операторы или выражения выполняются только при выполнении условия. Другими словами, операторы в фигурных скобках выполняются только тогда, когда логическое выражение в ifветке возвращает trueзначение.

Напишите ifутверждение для состояния красного светофора:

1. Внутри main()функции создайте trafficLightColorпеременную и присвойте ей "Red"значение:

fun main() {  
    val trafficLightColor = "Red"  
}

1. Добавьте ifоператор для состояния красного светофора, а затем передайте ему trafficLightColor == "Red"выражение:

fun main() {  
    val trafficLightColor = "Red"  
  
    if (trafficLightColor == "Red") {  
          
    }   
}

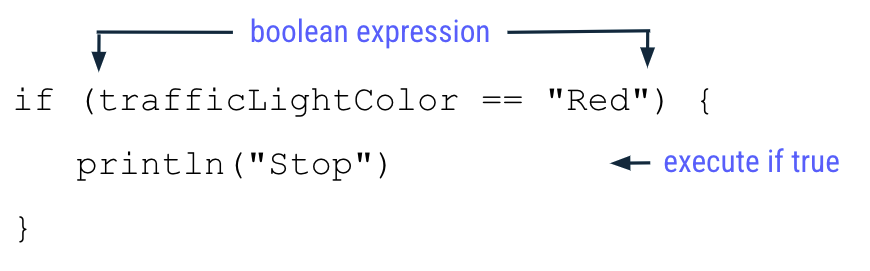
1. В теле инструкции ifдобавьте println()функцию, а затем передайте ей "Stop"аргумент:

fun main() {  
    val trafficLightColor = "Red"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    }   
}

1. Запустите программу и просмотрите результат:

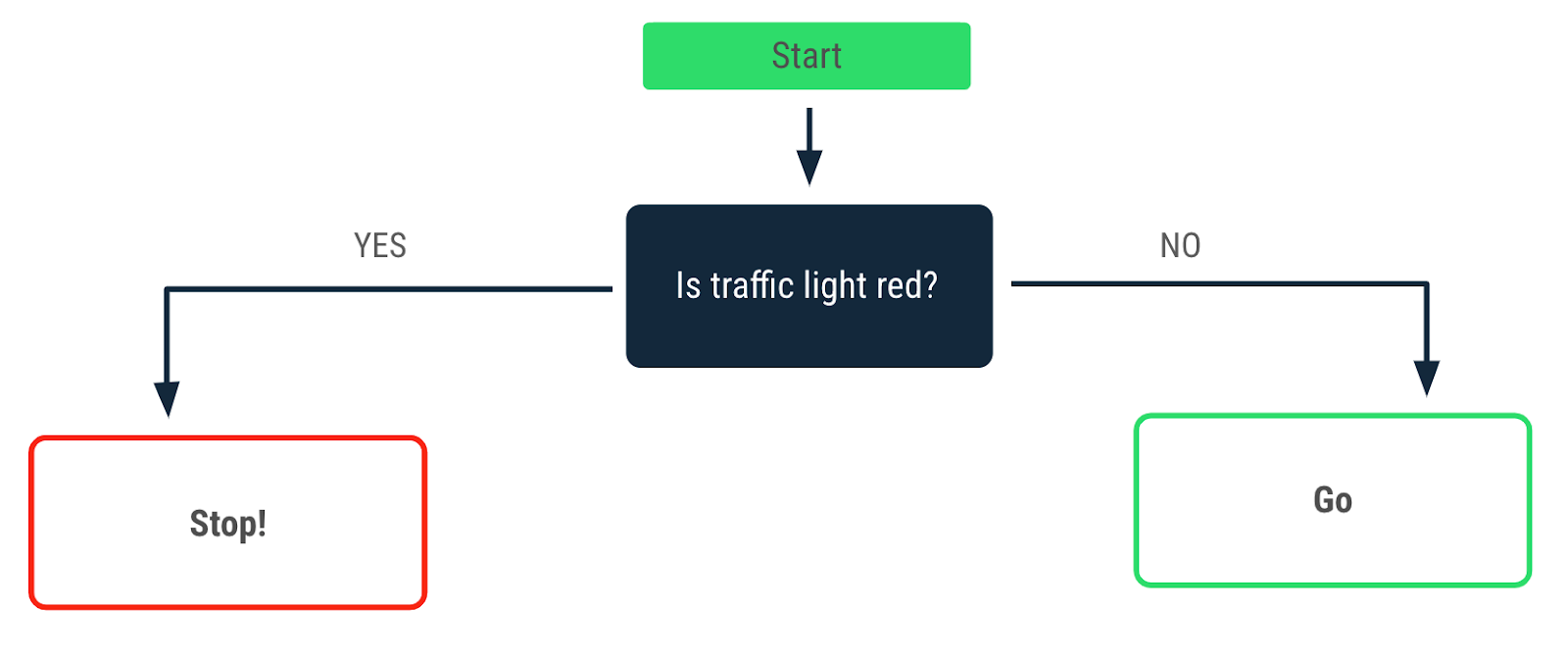
Stop

Выражение trafficLightColor == "Red"возвращает trueзначение, поэтому println("Stop")выполняется инструкция, которая печатает Stopсообщение.

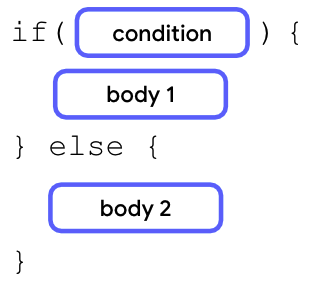


Добавить elseфилиал

Теперь вы можете расширить программу, чтобы она сообщала водителям ехать, *когда* светофор не горит красным.



Вам нужно добавить elseветку для создания if/elseоператора. Ветка — это незавершенная часть кода *,* которую можно объединить для формирования операторов или выражений. Ветка elseдолжна следовать за ifветкой.



После закрывающей фигурной скобки оператора ifвы добавляете elseключевое слово, за которым следует пара фигурных скобок. Внутри фигурных скобок оператора elseвы можете добавить второе тело, которое выполняется только тогда, когда условие в ifветке ложно.

Добавьте elseветку в вашу программу:

1. После закрывающей фигурной скобки оператора ifдобавьте elseключевое слово, за которым следует еще одна пара фигурных скобок:

fun main() {  
    val trafficLightColor = "Red"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else {  
  
    }  
}

1. Внутри elseфигурных скобок ключевого слова добавьте функцию println(), а затем передайте ей "Go"аргумент:

fun main() {  
    val trafficLightColor = "Red"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else {  
        println("Go")  
    }  
}

1. Запустите эту программу и просмотрите результат:

Stop

Программа по-прежнему ведет себя так же, как и до добавления elseветки, но не выводит Goсообщение.

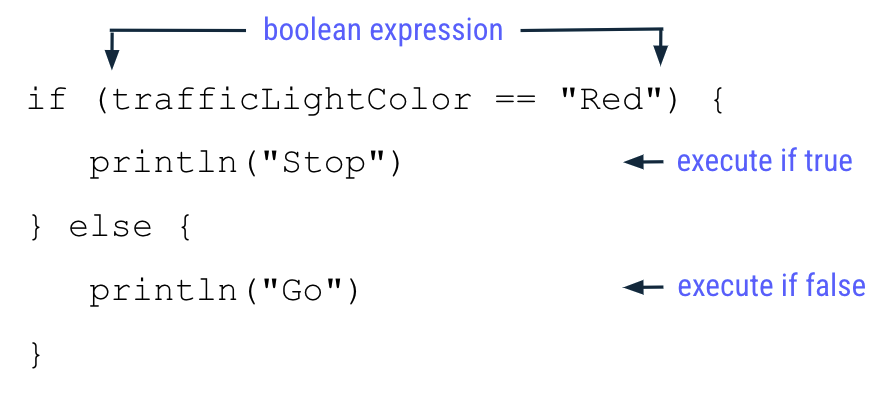
1. Присвойте trafficLightColorпеременной значение, "Green"потому что вы хотите, чтобы драйверы *светились* зеленым цветом:

fun main() {  
    val trafficLightColor = "Green"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else {  
        println("Go")  
    }  
}

1. Запустите эту программу и просмотрите результат:

Go

Как видите, теперь программа Goвместо сообщения печатает Stopсообщение.

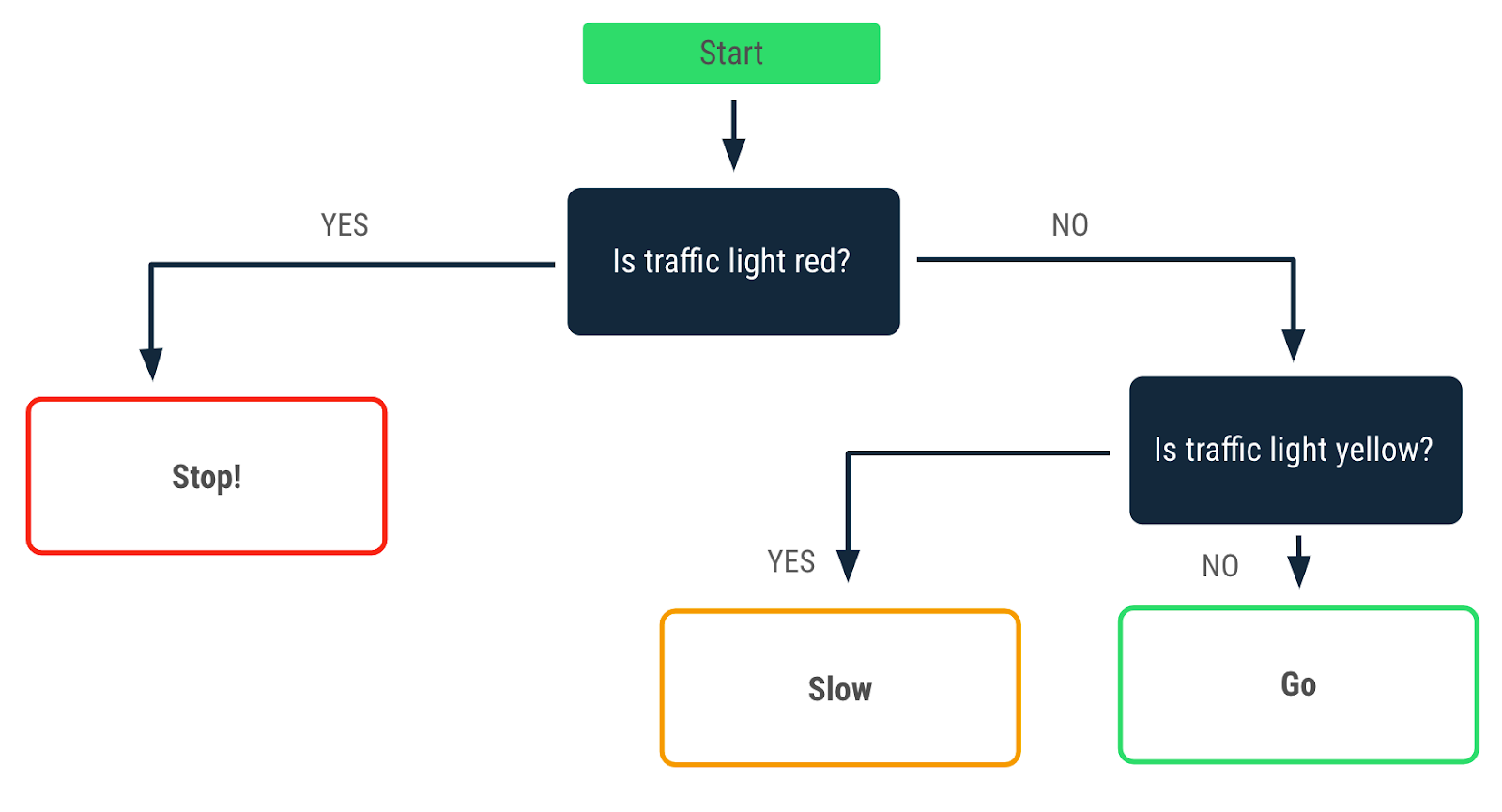


Вы переприсвоили trafficLightColorпеременной "Green"значение, поэтому trafficLightColor == "Red"выражение, оцененное в ifветке, возвращает falseзначение, поскольку это "Green"значение не равно значению "Red".

В результате программа пропускает все инструкции в ifветке и вместо этого выполняет все инструкции внутри elseветки. Это означает, что println("Go")функция выполняется, но сама println("Stop")функция не выполняется.

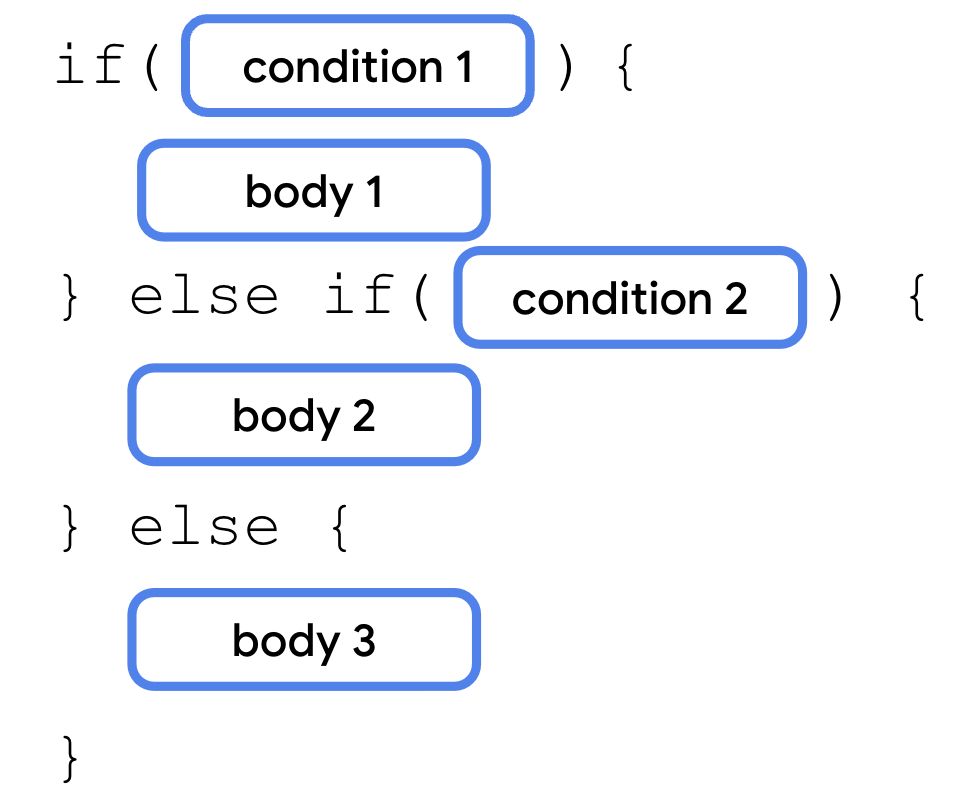
Добавить else ifфилиал

Обычно светофор также имеет желтый цвет, который сигнализирует водителям двигаться медленно. Вы можете расширить процесс принятия решений в программе, чтобы отразить это.

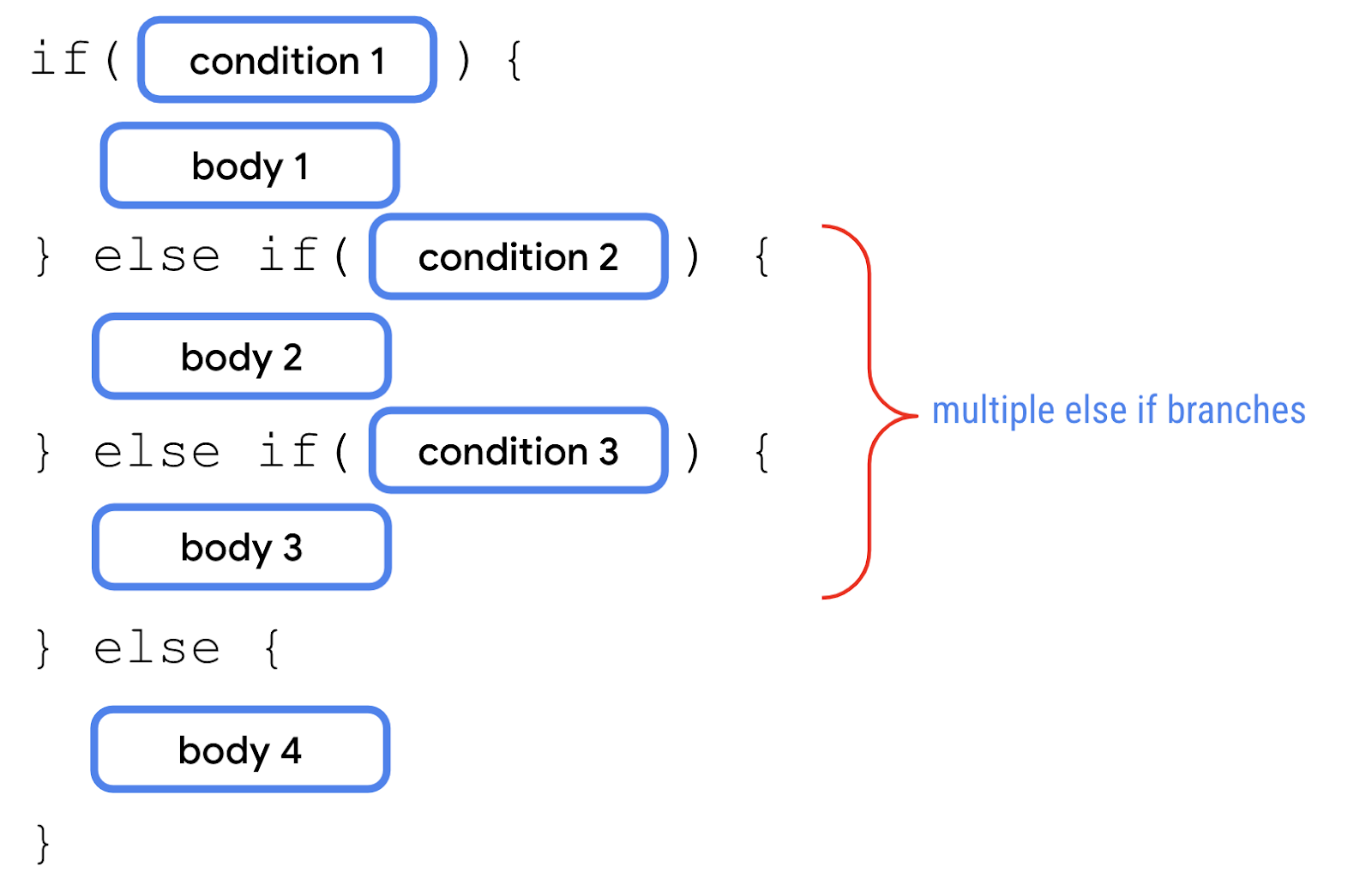


Вы научились писать условные выражения, учитывающие единственную точку принятия решения, с if/elseоператорами, содержащими одну ifи одну elseветвь. Как справиться с более сложным ветвлением с несколькими точками принятия решений? Когда вы сталкиваетесь с несколькими точками принятия решения, вам необходимо создать условные выражения с несколькими уровнями условий, что вы можете сделать, добавляя else ifветки к if/elseоператорам.

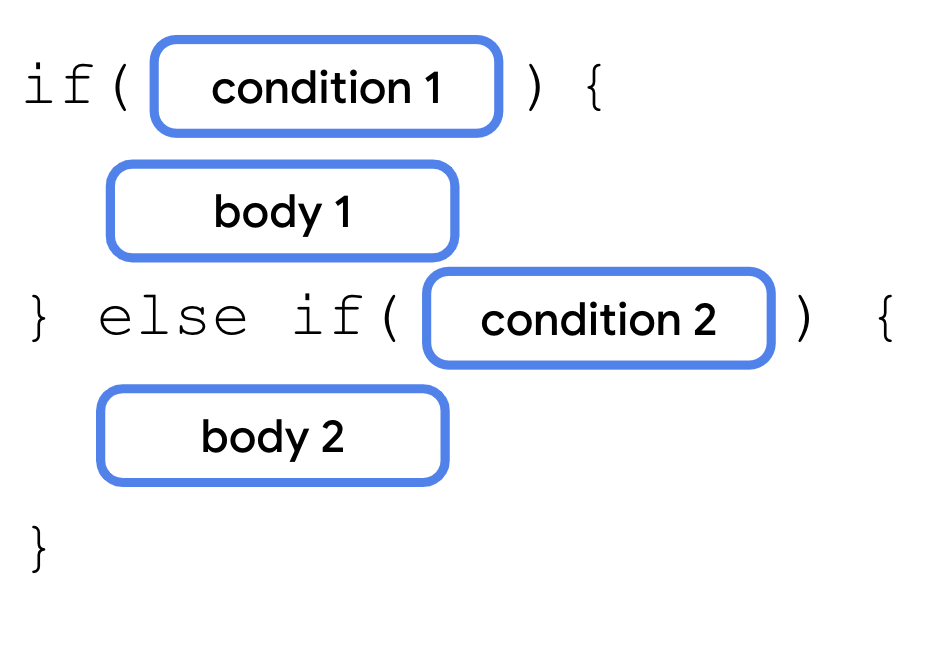
После закрывающей фигурной скобки ветки ifнужно добавить else ifключевое слово. Внутри круглых скобок ключевого else ifслова вам нужно добавить логическое выражение в качестве условия для else ifветки, за которым следует тело внутри пары фигурных скобок. Тело выполняется только в том случае, если условие 1 не выполнено, но условие 2 удовлетворено.



Ветка else ifвсегда располагается после ifветки, но перед elseветкой. Вы можете использовать несколько else ifветвей в операторе:



Оператор ifтакже может содержать ifветку и else ifветки без elseветки:



Добавьте else ifветку в вашу программу:

1. После закрывающей фигурной скобки оператора ifдобавьте else if (trafficLightColor == "Yellow")выражение, за которым следуют фигурные скобки:

fun main() {  
    val trafficLightColor = "Green"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
  
    } else {  
        println("Go")  
    }  
}

1. Внутри фигурных скобок ветки else ifдобавьте println()оператор, а затем передайте ему "Slow"строковый аргумент:

fun main() {  
    val trafficLightColor = "Green"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else {  
        println("Go")  
    }  
}

1. Переприсвойте trafficLightColorпеременной "Yellow"строковое значение:

fun main() {  
    val trafficLightColor = "Yellow"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else {  
        println("Go")  
    }  
}

1. Запустите эту программу и просмотрите результат:

Slow

Теперь программа печатает Slowсообщение вместо сообщения Stopили Go.



Вот почему он печатает только Slowсообщение, а не другие строки:

* Переменной trafficLightColorприсваивается "Yellow"значение.
* Значение "Yellow"не равно значению "Red", поэтому логическое выражение ветки if(обозначенное на изображении цифрой 1) возвращает значение false. Программа пропускает все операторы внутри ifветки и не выводит Stopсообщение.
* Поскольку ifветвь дает falseзначение, программа переходит к вычислению логического выражения внутри else ifветки.
* Значение "Yellow"равно значению "Yellow", поэтому логическое выражение ветки else if(обозначенное цифрой 2 на изображении) возвращает значение true. Программа выполняет все операторы внутри else ifветки и печатает Slowсообщение.
* Поскольку логическое выражение ветки else ifвозвращает trueзначение, программа пропускает остальные ветки. Таким образом, все операторы ветки elseне выполняются и программа не выводит Goсообщение.

Попробуй

Вы заметили, что текущая программа содержит ошибку?

В модуле 1 вы узнали о типе ошибки, называемой ошибкой *компиляции* , при которой Kotlin не может скомпилировать код из-за синтаксической ошибки в вашем коде, и программа не может работать. Здесь вы сталкиваетесь с другим типом ошибки, называемой *логической ошибкой* , при которой программа может работать, но не выдает ожидаемый результат.

Предположительно, вы хотите, чтобы водители ездили только тогда, когда цвет светофора зеленый. Что делать, если светофор сломался и выключился? Хотели бы вы, чтобы водитель вел машину или получил предупреждение о том, что что-то не так?

К сожалению, в текущей программе, если цвет светофора отличается *от* красного или желтого, водителю все равно рекомендуется *ехать* .

Исправьте эту проблему:

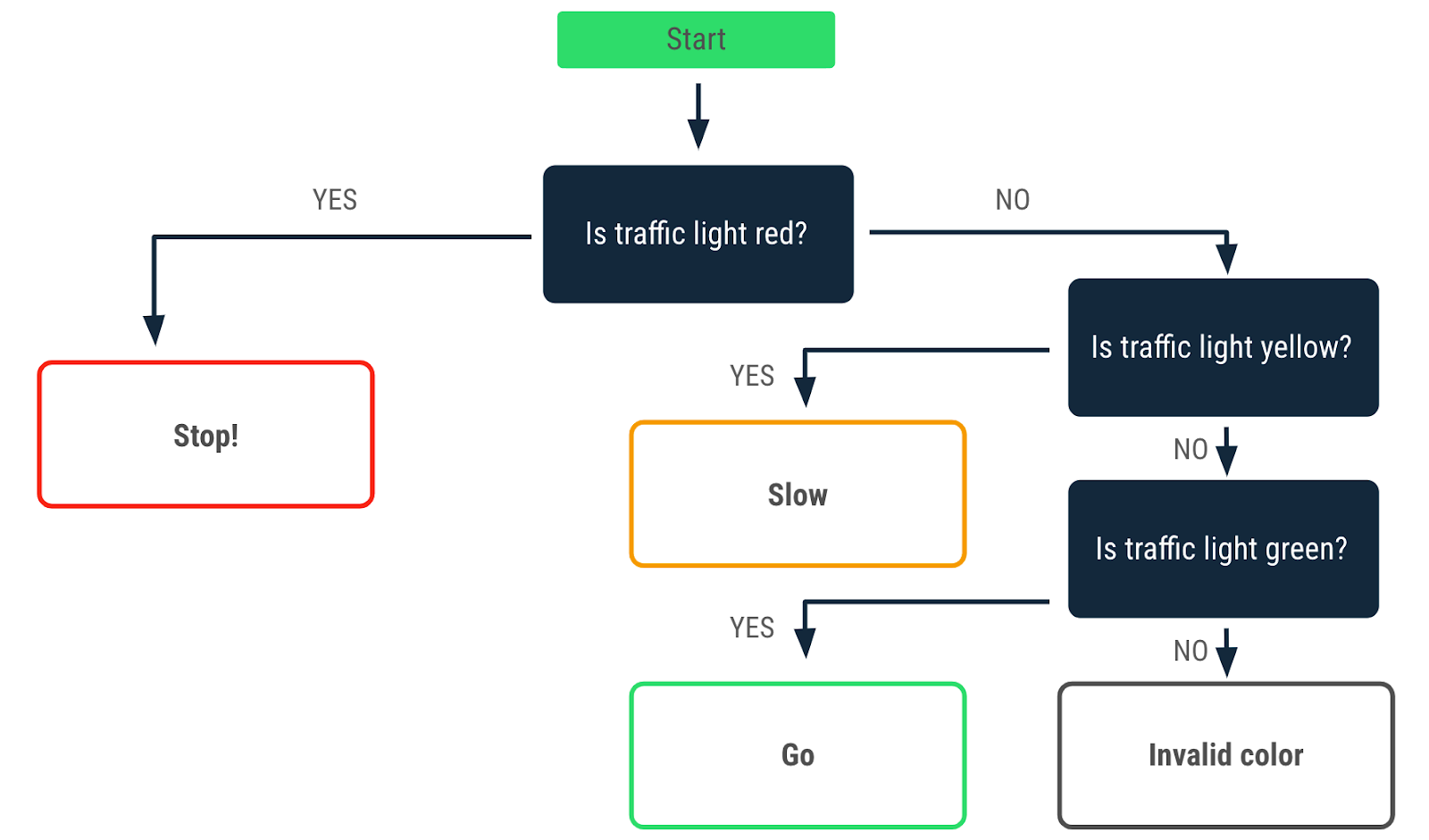
1. Присвойте trafficLightColorпеременной значение "Black", иллюстрирующее выключенный светофор:

fun main() {  
    val trafficLightColor = "Black"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else {  
        println("Go")  
    }  
}

1. Запустите эту программу и просмотрите результат:

Go

Обратите внимание, что программа печатает Goсообщение, даже если trafficLightColorпеременной не присвоено "Green"значение. Можете ли вы исправить эту программу, чтобы она отображала правильное поведение?



Вам необходимо изменить программу так, чтобы она печатала:

* Сообщение Goтолько тогда, когда trafficLightColorпеременной присвоено "Green"значение.
* Сообщение Invalid traffic-light color, когда trafficLightColorпеременной не присвоено значение "Red", "Yellow"или "Green"значение.

Исправить elseветку

Ветка elseвсегда располагается в конце оператора, if/elseпоскольку это универсальная ветка. Он автоматически выполняется, когда все остальные условия в предыдущих ветвях не удовлетворены. Таким образом, elseветвь не подходит, если вы хотите, чтобы действие выполнялось только тогда, когда оно удовлетворяет определенному условию. В случае со светофором вы можете использовать ветку else if, чтобы указать условие для зеленого света.

Используйте else ifветку для оценки состояния зеленого светофора:

1. После текущей else ifветки добавьте еще одну else if (trafficLightColor == "Green")ветку:

fun main() {  
    val trafficLightColor = "Black"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else if (trafficLightColor == "Green") {  
        println("Go")  
    }  
}

1. Запустите эту программу и просмотрите результат.

Вывод пуст, потому что у вас нет elseветки, которая выполняется, когда предыдущие условия не удовлетворены.

1. После последней else ifветки добавьте elseветку с println("Invalid traffic-light color")оператором внутри:

fun main() {  
    val trafficLightColor = "Black"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else if (trafficLightColor == "Green") {  
        println("Go")  
    } else {  
        println("Invalid traffic-light color")  
    }  
  
}

1. Запустите эту программу и просмотрите результат:

Invalid traffic-light color

1. Присвойте trafficLightColorпеременной другое значение, кроме "Red", "Yellow", или "Green", а затем перезапустите программу.

Каков вывод программы?

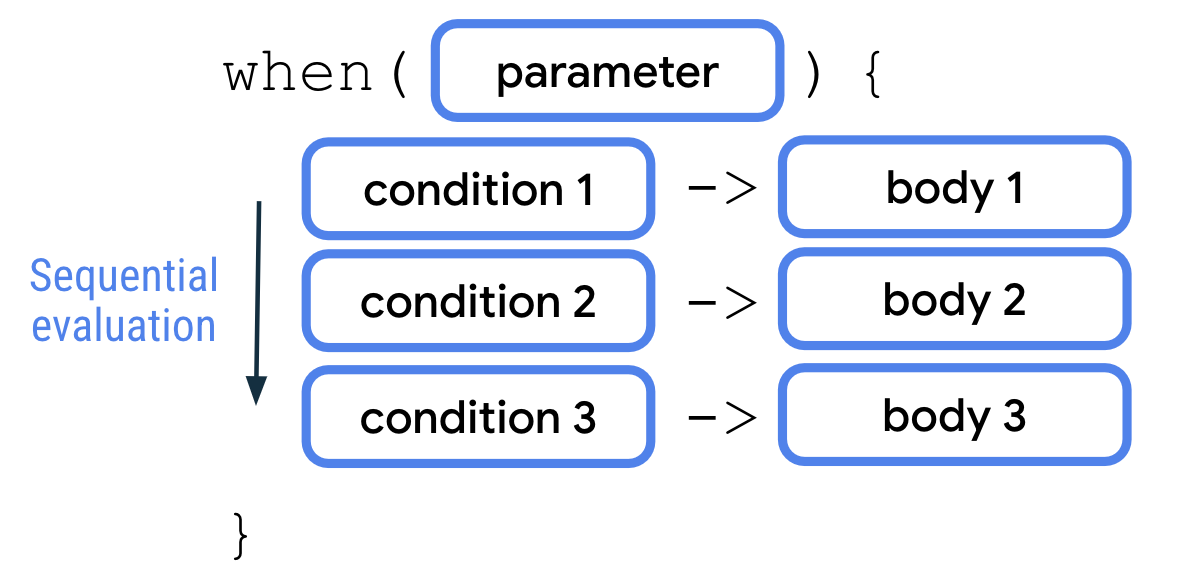
Хорошей практикой программирования является наличие явной else ifветки для проверки ввода зеленого цвета и elseветки для обнаружения других недопустимых входных данных. Это гарантирует, что водителям будет предложено *ехать* только тогда, когда загорится зеленый сигнал светофора. В других случаях передается явное сообщение о том, что светофор ведет себя не так, как ожидалось.

## [3. Используйте оператор if для нескольких ветвей.](https://developer.android.com/codelabs/basic-android-kotlin-compose-conditionals?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-conditionals#2)

Ваша trafficLightColorпрограмма выглядит более сложной с несколькими условиями, также известными как ветвление. Вы можете задаться вопросом, можно ли упростить программу с еще большим количеством ветвей.

В Kotlin, когда вы имеете дело с несколькими ветвями, вы можете использовать whenоператор вместо if/elseоператора, поскольку это улучшает читаемость, что означает, насколько легко читателям, обычно разработчикам, читать код. Очень важно учитывать читабельность при написании кода, поскольку вполне вероятно, что другим разработчикам придется просматривать и изменять ваш код на протяжении всей его жизни. Хорошая читаемость гарантирует, что разработчики смогут правильно понять ваш код и не допустить непреднамеренного внесения в него ошибок.

whenоператоры предпочтительнее, когда необходимо учитывать более двух ветвей.



Оператор whenпринимает одно значение через параметр. Затем значение последовательно оценивается по каждому из условий. Затем выполняется соответствующее тело первого выполненного условия. Каждое условие и тело разделены стрелкой ( ->). Подобно if/elseоператорам, каждая пара условия и тела называется ветвью whenоператоров. Аналогично оператору if/else, вы можете добавить elseветвь в качестве конечного условия в whenоператоре, который работает как всеобъемлющая ветвь.

## Перепишите if/elseвысказывание с whenутверждением

В программе «Светофор» уже есть несколько ветвей:

* Красный цвет светофора
* Желтый цвет светофора
* Зеленый цвет светофора
* Другой цвет светофора

Преобразуйте программу для использования whenоператора:

1. В main()функции удалите if/elseоператор:

fun main() {  
    val trafficLightColor = "Black"  
  
}

1. Добавьте whenоператор и затем передайте ему trafficLightColorпеременную в качестве аргумента:

fun main() {  
    val trafficLightColor = "Black"  
  
    when (trafficLightColor) {  
    }  
}

1. В текст оператора whenдобавьте "Red"условие, за которым следует стрелка и println("Stop")тело:

fun main() {  
    val trafficLightColor = "Black"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
    }  
}

1. В следующей строке добавьте "Yellow"условие, за которым следуют стрелка и println("Slow")тело:

fun main() {  
    val trafficLightColor = "Black"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow" -> println("Slow")  
    }  
}

1. В следующей строке добавьте "Green"условие, за которым следует стрелка, а затем тело println("Go"):

fun main() {  
    val trafficLightColor = "Black"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow" -> println("Slow")  
        "Green" -> println("Go")  
    }  
}

1. В следующей строке добавьте elseключевое слово, затем стрелку, а затем текст println("Invalid traffic-light color"):

fun main() {  
    val trafficLightColor = "Black"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}

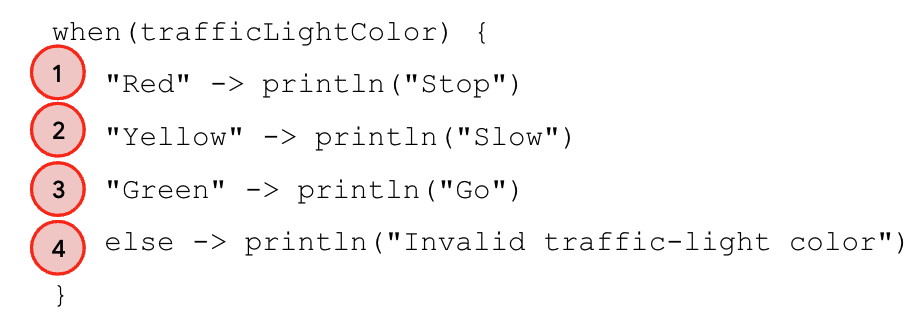
1. Переприсвойте trafficLightColorпеременной значение "Yellow":

fun main() {  
    val trafficLightColor = "Yellow"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}

Как вы думаете, какой будет результат, когда вы запустите эту программу?

1. Запустите программу и просмотрите результат:

Slow



Вывод представляет собой Slowсообщение, потому что:

* Переменной trafficLightColorприсваивается "Yellow"значение.
* Программа последовательно оценивает каждое условие.
* Значение "Yellow"не равно значению "Red", поэтому программа пропускает первое тело.
* Значение "Yellow"равно значению "Yellow", поэтому программа выполняет второе тело и печатает сообщение Slow.
* Тело было выполнено, поэтому программа игнорирует третью и четвертую ветви и оставляет whenоператор.

**Примечание.** Существует вариант оператора when, который не принимает никаких параметров и используется вместо цепочки if/else. Чтобы узнать больше, см. [выражение When](https://kotlinlang.org/docs/control-flow.html#when-expression) .

## Напишите более сложные условия в whenзаявлении

До сих пор вы научились писать whenусловия для одного условия равенства, например, когда trafficLightColorпеременной присваивается "Yellow"значение. Далее вы научитесь использовать запятую ( ,), inключевое слово и isключевое слово для формирования более сложных whenусловий.

Создайте программу, которая определяет, является ли число от 1 до 10 простым числом:

1. Откройте [игровую площадку Kotlin](https://developer.android.com/training/kotlinplayground) в отдельном окне.

К программе «Светофор» вы вернетесь позже.

1. Определите xпеременную, а затем присвойте ей значение 3:

fun main() {  
    val x = 3  
}

1. Добавьте whenоператор, включающий несколько ветвей для 2, и условий 3, и после каждого укажите тело :57println("x is prime number between 1 and 10.")

fun main() {  
    val x = 3  
  
    when (x) {  
        2 -> println("x is a prime number between 1 and 10.")  
        3 -> println("x is a prime number between 1 and 10.")  
        5 -> println("x is a prime number between 1 and 10.")  
        7 -> println("x is a prime number between 1 and 10.")  
    }  
}

1. Добавляем elseветку с println("x is not prime number between 1 and 10.")телом:

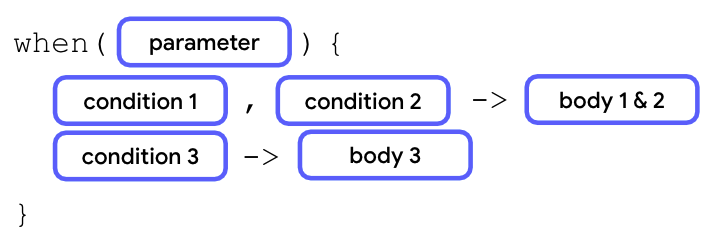
fun main() {  
    val x = 3  
  
    when (x) {  
        2 -> println("x is a prime number between 1 and 10.")  
        3 -> println("x is a prime number between 1 and 10.")  
        5 -> println("x is a prime number between 1 and 10.")  
        7 -> println("x is a prime number between 1 and 10.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. Запустите программу и затем убедитесь, что вывод соответствует ожиданиям:

x is a prime number between 1 and 10.

### **Используйте запятую ( ,) для нескольких условий**

Программа простых чисел содержит много повторений println()утверждений. При написании whenоператора вы можете использовать запятую ( ,), чтобы обозначить несколько условий, соответствующих одному и тому же телу.



На предыдущей диаграмме, если выполнено первое или второе условие, выполняется соответствующее тело.

Перепишите программу простых чисел, используя эту концепцию:

1. В ветке условия 2добавьте 3, 5разделив 7запятыми ( ,):

fun main() {  
    val x = 3  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        3 -> println("x is a prime number between 1 and 10.")  
        5 -> println("x is a prime number between 1 and 10.")  
        7 -> println("x is a prime number between 1 and 10.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. Удалите отдельные ветки для условий 3и :57

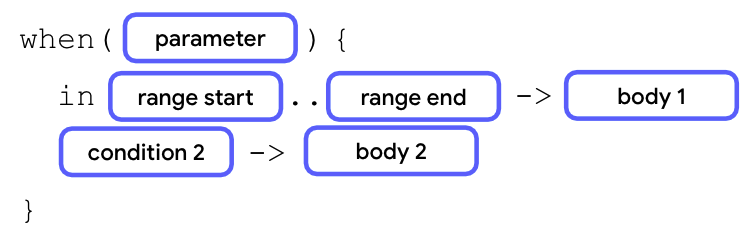
fun main() {  
    val x = 3  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. Запустите программу и затем убедитесь, что вывод соответствует ожиданиям:

x is a prime number between 1 and 10.

### **Используйте in ключевое слово для ряда условий**

Помимо символа запятой ( ,) для обозначения нескольких условий, вы также можете использовать inключевое слово и диапазон значений в whenветвях.



Чтобы использовать диапазон значений, добавьте число, обозначающее начало диапазона, за которым следуют две точки без пробелов, а затем закройте его другим числом, обозначающим конец диапазона.

Когда значение параметра равно любому значению в диапазоне между началом диапазона и концом диапазона, выполняется первое тело.

Можете ли вы в своей программе вычисления простых чисел напечатать сообщение, если число находится в диапазоне от 1 до 10, но не является простым числом?

Добавьте еще одну ветку с inключевым словом:

1. После первой ветки оператора whenдобавьте вторую ветку с inключевым словом, за которым следует 1..10диапазон и println("x is a number between 1 and 10, but not a prime number.")тело:

fun main() {  
    val x = 3  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. Измените xпеременную на 4значение:

fun main() {  
    val x = 4  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. Запустите программу и затем проверьте вывод:

x is a number between 1 and 10, but not a prime number.

Программа печатает сообщение второй ветки, но не сообщение первой или третьей ветки.

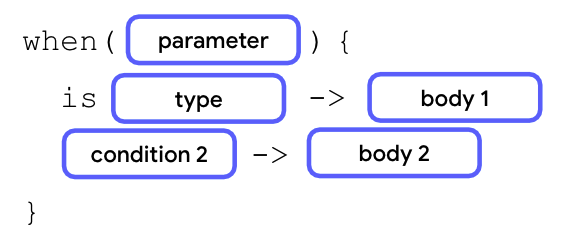
### 

Вот как работает эта программа:

* Переменной xприсваивается 4значение.
* Программа переходит к оценке условий для первой ветки. Значением 4не являются значения 2, 3, 5или 7, поэтому программа пропускает выполнение тела первой ветки и переходит ко второй ветке.
* Значение 4находится между 1и , поэтому печатается 10сообщение body.x is a number between 1 and 10, but not a prime number.
* Выполняется одно тело, поэтому программа покидает whenоператор и игнорирует elseветвь.

### **Используйте isключевое слово для проверки типа данных**

Вы можете использовать isключевое слово в качестве условия для проверки типа данных оцениваемого значения.



На предыдущей диаграмме, если значение аргумента имеет указанный тип данных, выполняется первое тело.

Можете ли вы в своей программе вычисления простых чисел напечатать сообщение, если входные данные представляют собой целое число, находящееся за пределами диапазона от 1 до 10?

Добавьте еще одну ветку с isключевым словом:

1. Измените x, чтобы он имел тип Any. Это означает, что оно xможет иметь значение, отличное от Intтипа.

fun main() {  
    val x: Any = 4  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. После второй ветки оператора whenдобавьте isключевое слово и Intтип данных с println("x is an integer number, but not between 1 and 10.")телом:

fun main() {  
    val x: Any = 4  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")  
        is Int -> println("x is an integer number, but not between 1 and 10.")  
        else -> println("x isn't a prime number between 1 and 10.")  
    }  
}

1. В elseветке меняем тело на println("x isn't an integer number.")тело:

fun main() {  
    val x: Any = 4  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")  
        is Int -> println("x is an integer number, but not between 1 and 10.")  
        else -> println("x isn't an integer number.")  
    }  
}

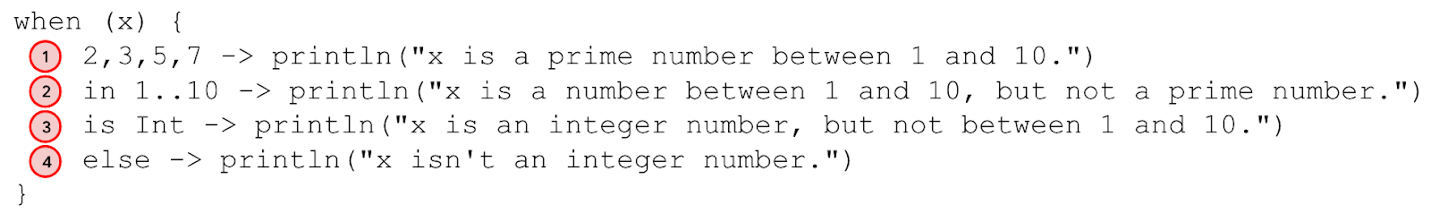
1. Измените xпеременную на 20значение:

fun main() {  
    val x: Any = 20  
  
    when (x) {  
        2, 3, 5, 7 -> println("x is a prime number between 1 and 10.")  
        in 1..10 -> println("x is a number between 1 and 10, but not a prime number.")  
        is Int -> println("x is an integer number, but not between 1 and 10.")  
        else -> println("x isn't an integer number.")  
    }  
}

1. Запустите программу и затем проверьте вывод:

x is an integer number, but not between 1 and 10.

Программа печатает сообщение третьей ветки, но не сообщения первой, второй или четвертой ветки.



Вот как работает программа:

* Переменной xприсваивается 20значение.
* Программа переходит к оценке условий для первой ветки. Значением 20не являются значения 2, 3, 5или 7, поэтому программа пропускает выполнение тела первой ветки и переходит ко второй ветке.
* Значение 20находится за пределами диапазона 1to 10, поэтому программа пропускает выполнение тела второй ветки и переходит к третьей ветке.
* Значение 20имеет Intтип, поэтому x is an integer number, but not between 1 and 10печатается тело.
* Выполняется одно тело, поэтому программа покидает whenоператор и игнорирует elseветвь.

## Попробуй

Теперь попрактикуйтесь в том, что вы узнали в программе «Светофор».

Представьте себе, что в некоторых странах светофор янтарного цвета предупреждает водителей так же, как желтый светофор в других странах. Можете ли вы изменить программу, чтобы учесть это дополнительное условие и сохранить исходные условия?

## Добавьте дополнительное условие с тем же телом

Добавьте в программу светофора дополнительное условие:

1. Если он у вас все еще открыт, вернитесь к экземпляру Kotlin Playground с вашей программой светофора.
2. Если вы закрыли его, откройте новый экземпляр Kotlin Playground и введите этот код:

fun main() {  
    val trafficLightColor = "Yellow"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}

1. Во второй ветви оператора whenдобавьте запятую после "Yellow"условия, а затем "Amber"условие:

fun main() {  
    val trafficLightColor = "Yellow"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow", "Amber" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}

1. Измените trafficLightColorпеременную на "Amber"значение:

fun main() {  
    val trafficLightColor = "Amber"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow", "Amber" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}

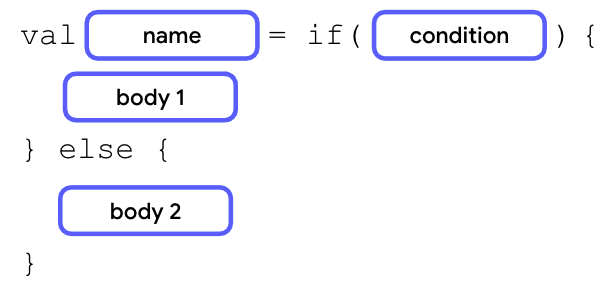
1. Запустите эту программу и затем проверьте вывод:

Slow

## [4. Используйте if/else и When в качестве выражений.](https://developer.android.com/codelabs/basic-android-kotlin-compose-conditionals?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-conditionals#3)

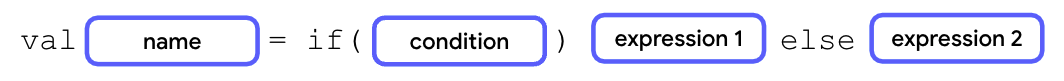
Вы узнали, как использовать if/elseи whenв качестве утверждений. Когда вы используете условные выражения в качестве операторов, вы позволяете каждой ветке выполнять различные действия в теле в зависимости от условий.

Вы также можете использовать условные выражения в качестве выражений для возврата разных значений для каждой ветви условия. Если тело каждой ветки похоже, вы можете использовать условные выражения, чтобы улучшить читаемость кода по сравнению с условными операторами.



Синтаксис условных выражений аналогичен операторам, но последняя строка тела в каждой ветви должна возвращать значение или выражение, а условные выражения присваиваются переменной.

Если тела содержат только возвращаемое значение или выражение, вы можете удалить фигурные скобки, чтобы сделать код более кратким.



В следующем разделе вы рассмотрите if/elseвыражения с помощью программы «Светофор».

## Преобразование ifоператора в выражение

println()В этом утверждении много повторов if/else:

fun main() {  
    val trafficLightColor = "Black"  
  
    if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else if (trafficLightColor == "Green") {  
        println("Go")  
    } else {  
        println("Invalid traffic-light color")  
    }  
  
}

Преобразуйте этот if/elseоператор в if/elseвыражение и удалите это повторение:

1. На игровой площадке Kotlin введите предыдущую программу светофора.
2. Определите messageпеременную, а затем присвойте ей if/elseоператор:

fun main() {  
    val trafficLightColor = "Black"  
  
    val message = if (trafficLightColor == "Red") {  
        println("Stop")  
    } else if (trafficLightColor == "Yellow") {  
        println("Slow")  
    } else if (trafficLightColor == "Green") {  
        println("Go")  
    } else {  
        println("Invalid traffic-light color")  
    }  
  
}

1. Удалите все println()операторы и их фигурные скобки, но оставьте значения внутри них:

fun main() {  
    val trafficLightColor = "Black"  
  
    val message =   
      if (trafficLightColor == "Red") "Stop"  
      else if (trafficLightColor == "Yellow") "Slow"  
      else if (trafficLightColor == "Green") "Go"  
      else "Invalid traffic-light color"  
}

1. Добавьте println()оператор в конец программы, а затем передайте ему переменную messageв качестве аргумента:

fun main() {  
    val trafficLightColor = "Black"  
  
    val message =   
      if (trafficLightColor == "Red") "Stop"  
      else if (trafficLightColor == "Yellow") "Slow"  
      else if (trafficLightColor == "Green") "Go"  
      else "Invalid traffic-light color"  
  
    println(message)  
}

1. Запустите эту программу и просмотрите результат:

Invalid traffic-light color

## Попробуй

Преобразуйте программу светофора, чтобы она использовала whenвыражение вместо whenоператора:

1. В Kotlin Playground введите этот код:

fun main() {  
    val trafficLightColor = "Amber"  
  
    when (trafficLightColor) {  
        "Red" -> println("Stop")  
        "Yellow", "Amber" -> println("Slow")  
        "Green" -> println("Go")  
        else -> println("Invalid traffic-light color")  
    }  
}

Можете ли вы преобразовать whenутверждение в выражение, чтобы не повторять println()утверждения?

1. Создайте messageпеременную и присвойте ее выражению when:

fun main() {  
    val trafficLightColor = "Amber"  
  
    val message = when(trafficLightColor) {  
        "Red" -> "Stop"  
        "Yellow", "Amber" -> "Slow"  
        "Green" -> "Go"  
        else -> "Invalid traffic-light color"  
    }  
}

1. Добавьте println()оператор в последнюю строку программы, а затем передайте ему переменную messageв качестве аргумента:

fun main() {  
    val trafficLightColor = "Amber"  
  
    val message = when(trafficLightColor) {  
        "Red" -> "Stop"  
        "Yellow", "Amber" -> "Slow"  
        "Green" -> "Go"  
        else -> "Invalid traffic-light color"  
    }  
    println(message)  
}

**Примечание.** Для оператора whenне требуется elseопределять ветвь. Однако в большинстве случаев выражению whenтребуется elseветвь, поскольку whenвыражение должно возвращать значение. Таким образом, компилятор Kotlin проверяет, являются ли все ветки исчерпывающими. Ветка elseгарантирует, что не возникнет сценария, в котором переменной не будет присвоено значение.

## Краткое содержание

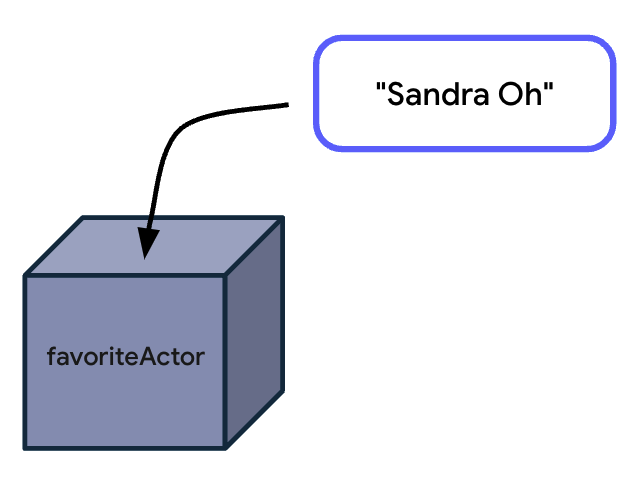
* В Котлине ветвление может быть достигнуто с помощью условных операторов if/elseили when.
* Тело ветки ifв if/elseусловии выполняется только тогда, когда логическое выражение внутри ifусловия ветвления возвращает trueзначение.
* Последующие else ifветки в if/elseусловии выполняются только тогда, когда предыдущие ifили else ifветки возвращают falseзначения.
* Последняя elseветвь в if/elseусловии выполняется только тогда, когда все предыдущие ifили else ifветви возвращают falseзначения.
* Рекомендуется использовать whenусловное выражение для замены if/elseусловного выражения, если имеется более двух ветвей.
* Вы можете написать более сложные условия в whenусловных выражениях с помощью запятой ( ,), inдиапазонов и isключевого слова.
* if/elseа whenусловные выражения могут работать как операторы или выражения.

## [2. Используйте переменные, допускающие значение NULL.](https://developer.android.com/codelabs/basic-android-kotlin-compose-nullability?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-nullability#1)

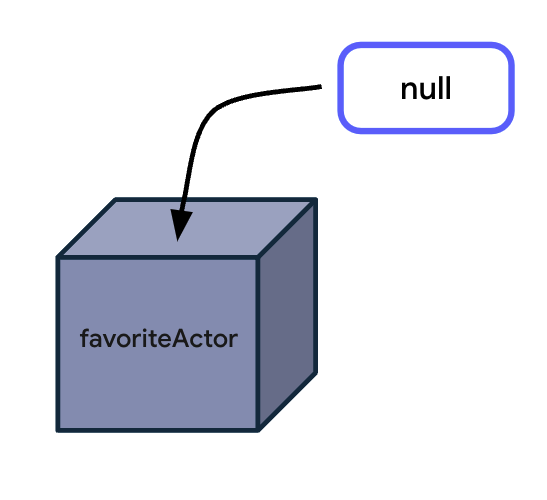
## Что null****?****

В модуле 1 вы узнали, что при объявлении переменной вам необходимо немедленно присвоить ей значение. Например, когда вы объявляете переменную favoriteActor, вы можете сразу же присвоить ей строковое значение "Sandra Oh".

val favoriteActor = "Sandra Oh"



Однако что делать, если у вас нет любимого актера? Возможно, вы захотите присвоить переменной a значение "Nobody"или "None". Это не очень хороший подход, потому что ваша программа интерпретирует переменную favoriteActor как имеющую значение "Nobody" или "None", а не как не имеющую значения вообще. В Котлине вы можете использовать его null, чтобы указать, что с переменной не связано никакого значения.



Чтобы использовать null в коде, выполните следующие действия:

1. В [Kotlin Playground](https://developer.android.com/training/kotlinplayground" \t "_blank) замените содержимое тела функции main() переменной, favoriteActor установленной в значение null:

fun main() {  
    val favoriteActor = null  
}

1. Распечатайте значение переменной favoriteActorс помощью println()функции, а затем запустите эту программу:

fun main() {  
    val favoriteActor = null  
    println(favoriteActor)  
}

Вывод выглядит следующим образом:

null

## Переназначение переменных с null

Ранее вы узнали, что переменные, определенные с помощью ключевого слова, можно переназначать var различным значениям одного и того же типа. Например, вы можете переназначить name переменную, объявленную с одним именем, другому имени, если новое имя имеет  тип String.

var favoriteActor: String = "Sandra Oh"  
favoriteActor = "Meryl Streep"

Бывают случаи, когда после объявления переменной вы можете захотеть присвоить эту переменную null. Например, после того, как вы объявили своего любимого актера, вы решаете, что вообще не хотите раскрывать информацию о своем любимом актере. В этом случае полезно присвоить favoriteActor переменную null.

## ****Понимание переменных, допускающих и не допускающих значение NULL.****

Чтобы переназначить favoriteActor переменную null, выполните следующие действия:

1. Измените val ключевое слово на var ключевое слово, а затем укажите, что favoriteActor переменная является типом String, и присвойте ей имя вашего любимого актера:

fun main() {  
    var favoriteActor: String = "Sandra Oh"  
    println(favoriteActor)  
}

1. Удалить println()функцию:

fun main() {  
    var favoriteActor: String = "Sandra Oh"  
}

1. Переназначьте favoriteActor переменную null и запустите эту программу:

fun main() {  
    var favoriteActor: String = "Sandra Oh"  
    favoriteActor = null  
}

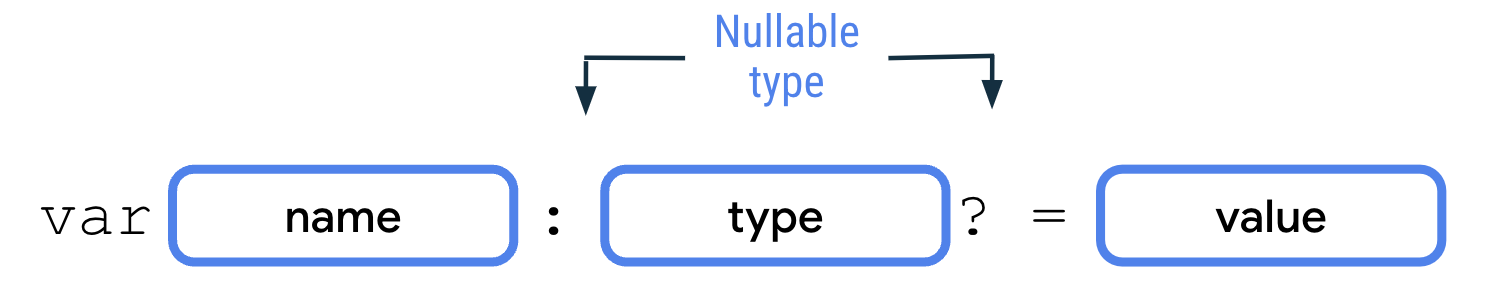
Вы получаете это сообщение об ошибке:



В Котлине существует различие между типами, допускающими и не допускающими значение NULL:

* Типы, допускающие значение NULL, — это переменные, которые могут содержать null.
* Ненулевые типы — это переменные, которые не могут содержать null.

Тип может принимать значение NULL только в том случае, если вы явно разрешили ему сохранять значение null. Как говорится в сообщении об ошибке,  тип данных String не допускает значения NULL, поэтому вы не можете переназначить переменную null.



Чтобы объявить переменные, допускающие значение NULL, в Kotlin, вам нужно добавить оператор ? в конец типа.  Например, String?тип может содержать либо строку, либо null, тогда как Stringтип может содержать только строку. Чтобы объявить переменную, допускающую значение NULL, вам необходимо явно добавить тип, допускающий значение NULL. Без типа, допускающего значение NULL, компилятор Kotlin делает вывод, что это тип, не допускающий значения NULL.

1. Измените favoriteActorтип переменной с Stringтипа данных на String?тип данных:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
    favoriteActor = null  
}

1. Распечатайте переменную favoriteActor до и после null переназначения, а затем запустите эту программу:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
    println(favoriteActor)  
  
    favoriteActor = null  
    println(favoriteActor)  
}

Вывод выглядит следующим образом:

Sandra Oh

null

Первоначально переменная favoriteActor содержала строку, а затем ей переназначалась null.

## ****Попробуй****

Теперь, когда вы можете использовать тип String?, допускающий значение NULL, можете ли вы инициализировать переменную значением Int и переназначить ее null?

## Запишите Int значение, допускающее значение NULL.

1. Удалите весь код в main()функции:

fun main() {  
      
}

1. Создайте переменную number типа Int, допускающую значение NULL, а затем присвойте ей значение 10:

fun main() {  
    var number: Int? = 10  
}

1. Распечатайте numberпеременную и затем запустите эту программу:

fun main() {  
    var number: Int? = 10  
    println(number)  
}

Результат ожидаемый:

10

1. Переназначьте numberпеременную, чтобы nullподтвердить, что переменная имеет значение NULL:

fun main() {  
    var number: Int? = 10  
    println(number)  
      
    number = null  
}

1. Добавьте еще один println(number)оператор в последнюю строку программы и запустите ее:

fun main() {  
    var number: Int? = 10  
    println(number)  
      
    number = null  
    println(number)  
}

Результат ожидаемый:

10

null

**Примечание.**  Используйте NULLable переменные, тогда, когда переменная действительно может получить значение null. И используйте обычные типы данных в ситуациях, когда значение Null не будет появляться. Так как доступ к переменным, допускающим значение NULL, требует более сложной обработки.

## [3. Обработка переменных, допускающих значение NULL.](https://developer.android.com/codelabs/basic-android-kotlin-compose-nullability?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-nullability#2)

Ранее вы научились использовать  оператор «.» для доступа к методам и свойствам переменных, не допускающих значения NULL. В этом разделе вы узнаете, как использовать его для доступа к методам и свойствам переменных, допускающих значение NULL.

Чтобы получить доступ к свойству переменной favoriteActor, не допускающей значения NULL, выполните следующие действия:

1. Удалите весь код в функции main(), а затем объявите  переменную favoriteActor типа String и присвойте ей имя любимого актера:

fun main() {  
    var favoriteActor: String = "Sandra"  
}

1. Распечатайте количество символов в значении переменной favoriteActor со свойством [length](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/length.html), а затем запустите эту программу:

fun main() {  
    var favoriteActor: String = "Sandra"  
    println(favoriteActor.length)  
}

Результат ожидаемый:

9

В значении переменной девять символов , включая favoriteActor  пробелы. Количество символов в имени вашего любимого актера может быть разным.

## Доступ к свойству переменной, допускающей значение NULL.

Представьте, что вы хотите сделать favoriteActorпеременную обнуляемой, чтобы люди, у которых нет любимого актера, могли присвоить эту переменную null.

Чтобы получить доступ к свойству favoriteActorпеременной, допускающей значение NULL, выполните следующие действия:

* Измените favoriteActorтип переменной на тип, допускающий значение NULL, а затем запустите эту программу:

fun main() {  
    var favoriteActor: String? = "Sandra"  
    println(favoriteActor.length)  
}

Вы получаете это сообщение об ошибке:



Эта ошибка является ошибкой компиляции . Как упоминалось в предыдущей лаборатории кода, ошибка компиляции возникает, когда Kotlin не может скомпилировать код из-за синтаксической ошибки в вашем коде.

Kotlin намеренно применяет синтаксические правила, чтобы обеспечить безопасность null, которая означает гарантию отсутствия случайных вызовов потенциальных null переменных. Это не означает, что переменные не могут быть null. Это означает, что если осуществляется доступ к члену переменной, переменная не может быть null.

Это очень важно, поскольку при попытке доступа к члену переменной, которая nullназывается null ссылкой, во время работы приложения, приложение аварийно завершает работу, поскольку переменная null не содержит какого-либо свойства или метода. Этот тип сбоя известен как ошибка времени выполнения , при которой ошибка возникает после компиляции и запуска кода.

Благодаря null безопасности Kotlin такие ошибки во время выполнения предотвращаются, поскольку компилятор Kotlin принудительно проверяет null типы , допускающие значение NULL. Null проверка относится к процессу проверки того, может ли переменная быть такой, nullпрежде чем к ней будет осуществлен доступ и она будет обработана как тип, не допускающий значения NULL. Если вы хотите использовать значение, допускающее значение NULL, в качестве типа, не допускающего NULL, вам необходимо null явно выполнить проверку. Вы узнаете об этом в разделе **«Использование**if/else **условий»** далее в этой лаборатории кода.

В этом примере происходит сбой кода во время компиляции, поскольку прямая ссылка на lengthсвойство переменной favoriteActorне разрешена, поскольку существует вероятность того, что переменная равна null.

Далее вы изучите различные методы и операторы для работы с типами, допускающими значение NULL.

## Используйте оператора безопасного вызова «?.».

Вы можете использовать ?.оператор безопасного вызова для доступа к методам или свойствам переменных, допускающих значение NULL.

## Диаграмма, на которой показан блок переменных, допускающий значение NULL, за которым следуют вопросительный знак, точка и блок метода или свойства. Между ними нет пробелов.

Чтобы использовать оператор безопасного вызова «?.» для доступа к методу или свойству, добавьте «?» после имени переменной и получите доступ к методу или свойству с помощью «.».

Оператор безопасного вызова ?. обеспечивает более безопасный доступ к переменным, допускающим значение NULL, поскольку компилятор Kotlin останавливает любую попытку доступа к ссылкам null и возвращает доступ null к члену, к которому осуществляется доступ.

Чтобы безопасно получить доступ к свойству переменной favoriteActor, допускающей значение NULL, выполните следующие действия:

1. В println()операторе замените .оператор на ?.оператор безопасного вызова:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
    println(favoriteActor?.length)  
}

1. Запустите эту программу и убедитесь, что вывод соответствует ожиданиям:

9

Количество символов в имени вашего любимого актера может отличаться.

1. Переназначьте favoriteActorпеременную null и запустите эту программу:

fun main() {  
    var favoriteActor: String? = null  
    println(favoriteActor?.length)  
}

Вы видите этот вывод:

null

Обратите внимание, программа не выполнила аварийного завершения работы, несмотря на обращение к свойству length переменной, имеющей значение Null. Оператор безопасного вызова просто возвращает null.

**Примечание.** Вы также можете использовать операторы безопасного вызова ?. для переменных, не допускающих значения NULL, для доступа к методу или свойству. Хотя компилятор Kotlin не выдаст при этом никаких ошибок, в этом нет необходимости, поскольку доступ к методам или свойствам для переменных, не допускающих значения NULL, всегда безопасен.

## Используйте  ненулевой оператор утверждения !!

Вы также можете использовать !!оператор утверждения not-null для доступа к методам или свойствам переменных, допускающих значение null.



После переменной, допускающей значение NULL, необходимо добавить оператор утверждения !!, не имеющий значения NULL, за которым следует  оператор ., а затем метод или свойство без каких-либо пробелов.

Как следует из названия, если вы используете утверждение not-null !!, это означает, что вы утверждаете, что значение переменной не равно null, независимо от того, так это или нет.

В отличие от операторов безопасного вызова ?., использование оператора утверждения !!, может привести к  выдаче ошибки NullPointerException, если переменная, допускающая значение NULL, действительно равна null. Таким образом, это следует делать только в том случае, если переменная всегда не допускает значения NULL или если установлена ​​правильная обработка исключений. Если исключения не обрабатываются, они вызывают ошибки во время выполнения.

Чтобы получить доступ к свойству переменной favoriteActor с помощью оператора утверждения !!, не являющегося нулевым, выполните следующие действия:

1. Переназначьте favoriteActorпеременную имени вашего любимого актера, а затем замените ?.оператор безопасного вызова ненулевым !!оператором утверждения в println()инструкции:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
    println(favoriteActor!!.length)  
}

1. Запустите эту программу и убедитесь, что вывод соответствует ожиданиям:

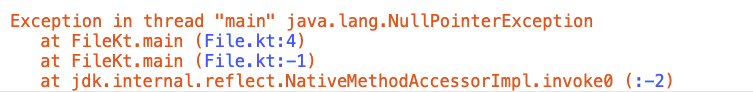
9

Количество символов в имени вашего любимого актера может отличаться.

1. Переназначьте favoriteActorпеременную nullи запустите эту программу:

fun main() {  
    var favoriteActor: String? = null  
    println(favoriteActor!!.length)  
}

Вы получаете ошибку NullPointerException:



Эта ошибка Kotlin показывает, что ваша программа потерпела крах во время выполнения. Таким образом, не рекомендуется использовать !!оператор утверждения ненулевого значения, если вы не уверены, что переменная не равна null.

**Примечание.** В других языках программирования, которые не содержат атрибут нулевой безопасности, ошибка  NullPointerException является частой причиной сбоев приложения. Таким образом, Kotlin устраняет серьезную причину сбоев программ, поскольку обеспечивает nullбезопасность языка.

## Используйте условные предложения if/else

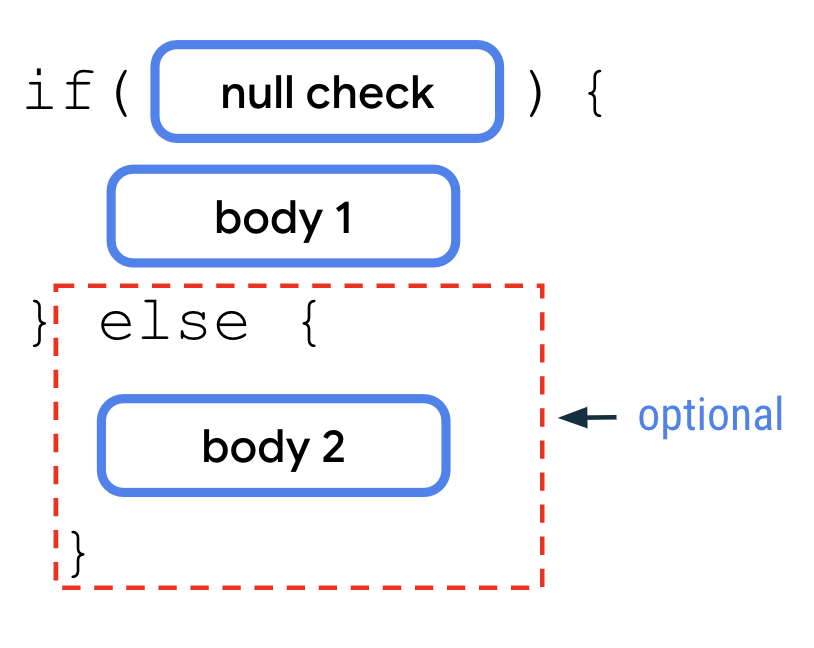
Вы можете использовать ветвь if в условиях if/else для выполнения null проверок .



Для выполнения null проверок вы можете убедиться, что переменная, допускающая значение NULL, не равна null с помощью оператора сравнения !=.

### **if/else заявления**

Оператор if/else можно использовать вместе с null проверкой следующим образом:



Проверка нуля полезна в сочетании с оператором if/else:

* В качестве условия используется проверка выражения if nullableVariable != null
* Тело 1 внутри if ветки предполагает, что переменная не допускает значения NULL. Таким образом, в этом теле вы можете свободно получать доступ к методам или свойствам переменной, как если бы это переменная, не допускающая значения NULL, без использования оператора ?.безопасного вызова или !!оператора утверждения, не имеющего значения NULL.
* Тело 2 внутри elseветки предполагает, что переменная равна null. Таким образом, в этом теле вы можете добавлять операторы, которые должны выполняться, когда переменная равна null. Ветка elseне является обязательной. ifДля запуска проверки можно использовать только условное выражение null, не указывая действие по умолчанию в случае nullсбоя проверки.

Проверку nullудобнее использовать с ifусловием, когда имеется несколько строк кода, использующих переменную, допускающую значение NULL. Напротив, ?.оператор безопасного вызова более удобен для одной ссылки на переменную, допускающую значение NULL.

Чтобы написать if/elseоператор с nullпроверкой переменной favoriteActor, выполните следующие действия:

1. Снова присвойте favoriteActorпеременной имя вашего любимого актера, а затем удалите println()оператор:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
}

1. Добавьте if ветку с favoriteActor != null условием:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    if (favoriteActor != null) {  
  
    }  
}

1. В теле ветки ifдобавьте printlnоператор, принимающий "The number of characters in your favorite actor's name is ${favoriteActor.length}."строку, а затем запустите эту программу:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    if (favoriteActor != null) {  
      println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")  
    }  
}

Результат соответствует ожиданиям.

The number of characters in your favorite actor's name is 9.

Количество символов в имени вашего любимого актера может отличаться.

Обратите внимание, что вы можете получить доступ к методу длины имени напрямую с помощью .оператора, поскольку вы получаете доступ к lengthметоду внутри ifветки после nullпроверки. favoriteActorТаким образом, компилятор Kotlin знает, что переменная не имеет значения null, поэтому компилятор разрешает прямой доступ к свойству.

1. Необязательно: добавьте elseветку для обработки ситуации, в которой имя актера null:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    if (favoriteActor != null) {  
      println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")  
    } else {  
  
    }  
}

1. В теле ветки elseдобавьте printlnоператор, принимающий "You didn't input a name."строку:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    if (favoriteActor != null) {  
      println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")  
    } else {  
      println("You didn't input a name.")  
    }  
}

1. Назначьте favoriteActorпеременную nullи запустите эту программу:

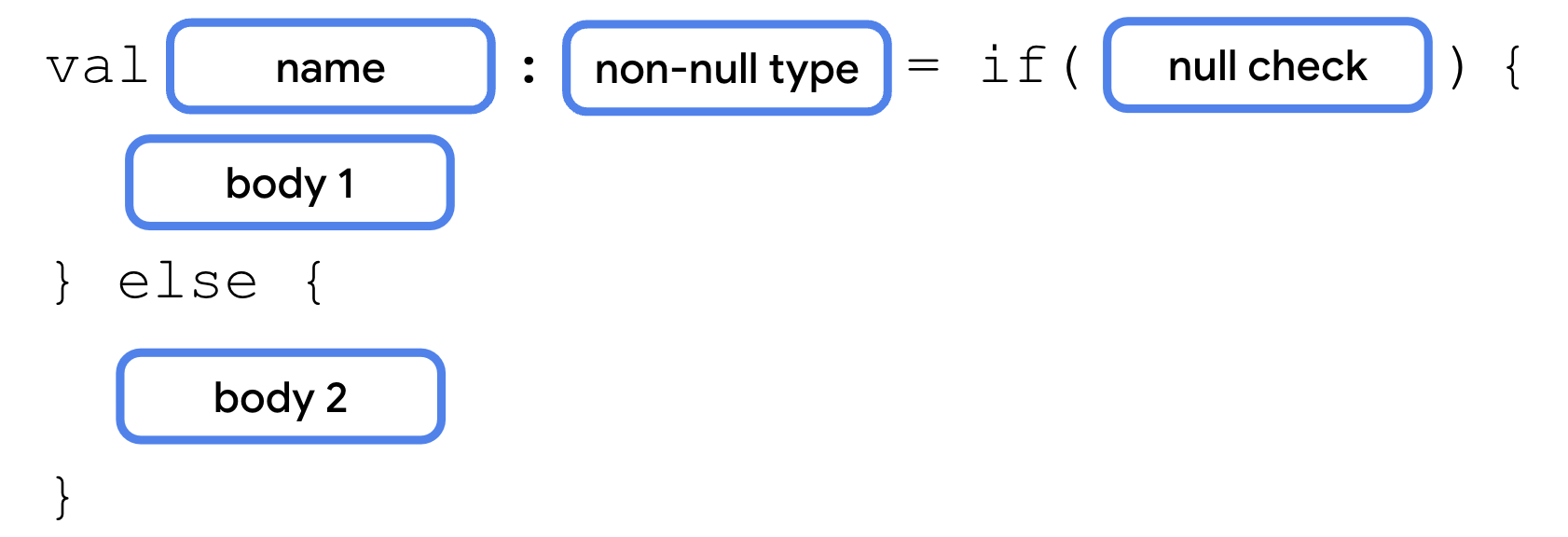
fun main() {  
    var favoriteActor: String? = null  
  
    if(favoriteActor != null) {  
      println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")  
    } else {  
      println("You didn't input a name.")  
    }  
}

Результат ожидаемый:

You didn't input a name.

### **if/else выражения**

Вы также можете объединить проверку null с  выражением if/else для преобразования переменной, допускающей значение NULL, в переменную, допускающую значение NULL.



Чтобы присвоить if/else выражение типу, не допускающему значения NULL:

* nullableVariable != null В качестве условия используется чек if.
* Тело 1 внутри ifветки предполагает, что переменная не допускает значения NULL. Таким образом, в этом теле вы можете получить доступ к методам или свойствам переменной, как если бы это переменная, не допускающая значения NULL, без ?.безопасного оператора или !!оператора утверждения, не имеющего значения NULL.
* Тело 2 внутри elseветки предполагает, что переменная равна null. Таким образом, в это тело вы можете добавлять операторы, которые должны выполняться, когда переменная равна null.
* В последней строке тела 1 и 2 вам нужно использовать выражение или значение, которое приводит к типу, не допускающему значения NULL, чтобы оно присваивалось переменной, не допускающей значение NULL, когда проверка null проходит или завершается неудачей соответственно.

Чтобы использовать if/elseвыражение для переписывания программы так, чтобы она использовала только один printlnоператор, выполните следующие действия:

1. Присвойте favoriteActorпеременной имя вашего любимого актера:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    if (favoriteActor != null) {  
      println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")  
    } else {  
      println("You didn't input a name.")  
    }  
}

1. Создайте lengthOfNameпеременную и затем присвойте ее выражению if/else:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = if (favoriteActor != null) {  
      println("The number of characters in your favorite actor's name is ${favoriteActor.length}.")  
    } else {  
      println("You didn't input a name.")  
    }  
}

1. Удалите оба println()оператора из ветвей ifи else:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = if (favoriteActor != null) {  
        
    } else {  
        
    }  
}

1. В тело ветки ifдобавьте favoriteActor.lengthвыражение:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = if (favoriteActor != null) {  
      favoriteActor.length  
    } else {  
        
    }  
}

Доступ к свойству length переменной favoriteActor осуществляется  непосредственно с помощью .оператора.

1. В теле ветки elseдобавьте 0значение:

fun main() {  
   var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = if (favoriteActor != null) {  
      favoriteActor.length  
    } else {  
      0  
    }  
}

Это 0значение служит значением по умолчанию, если имя равно null.

1. В конце функции main()добавьте printlnоператор, принимающий "The number of characters in your favorite actor's name is $lengthOfName."строку, а затем запустите эту программу:

**fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = if (favoriteActor != null) {  
      favoriteActor.length  
    } else {  
      0  
    }  
  
    println("The number of characters in your favorite actor's name is $lengthOfName.")**  
}

Результат ожидаемый:

The number of characters in your favorite actor's name is 9.

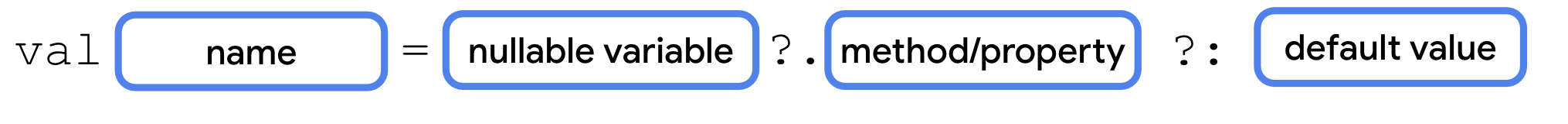
Количество символов в имени, которое вы использовали, может отличаться.

**Примечание.** Вместо оператора ==для проверок можно также использовать оператор сравнения . При этом обратите внимание, что два тела перевернуты. В теле ветки предполагается, что переменная равна нулю, а в теле ветки предполагается, что переменная не допускает значения NULL.null!=ifnullelse

## Используйте оператор Элвиса ?:

Оператор Элвиса ?:— это оператор, который можно использовать вместе с оператором безопасного вызова ?.. С помощью оператора Элвиса ?: вы можете добавить значение по умолчанию, когда оператор безопасного вызова ?. возвращает значение null.

Если переменная не равна null, выполняется выражение перед оператором ?: Элвиса. Если переменная равна null , выполняется выражение после оператора Элвиса ?:.



Чтобы изменить предыдущую программу для использования ?:оператора Элвиса, выполните следующие действия:

1. Удалите if/else условное выражение, затем присвойте lengthOfName переменной значение, допускающее значение NULL, favoriteActor и используйте ?.оператор безопасного вызова для вызова ее length свойства:

fun main() {  
   var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = favoriteActor?.length  
  
    println("The number of characters in your favorite actor's name is $lengthOfName.")  
}

1. После  свойства length добавьте ?:оператор Элвиса, затем 0 значение, а затем запустите эту программу:

fun main() {  
    var favoriteActor: String? = "Sandra Oh"  
  
    val lengthOfName = favoriteActor?.length ?: 0  
  
    println("The number of characters in your favorite actor's name is $lengthOfName.")  
}

Вывод такой же, как и предыдущий вывод:

The number of characters in your favorite actor's name is 9.

**Примечание.** Оператор «Элвис» ?: назван в честь [Элвиса Пресли](https://en.wikipedia.org/wiki/Elvis_Presley) , рок-звезды, поскольку , если смотреть на него сбоку, он напоминает смайлик в виде его [причёски .](https://en.wikipedia.org/wiki/Quiff)

Краткое содержание

* Переменной можно установить значение null, чтобы указать, что она не имеет значения.
* Переменные, не допускающие значения NULL, не могут быть назначены null.
* Могут быть назначены переменные, допускающие значение NULL null.
* Для доступа к методам или свойствам переменных, допускающих значение NULL, необходимо использовать ?.операторы безопасного вызова или !!операторы утверждения, не имеющие значения NULL.
* Вы можете использовать if/elseоператоры с nullпроверками для доступа к переменным, допускающим значение NULL, в контекстах, не допускающих значение NULL.
* Вы можете преобразовать переменную, допускающую значение NULL, в тип, не допускающий значения NULL, с помощью if/elseвыражений.
* Вы можете указать значение по умолчанию, если переменная, допускающая значение NULL, находится nullв if/elseвыражении или ?:операторе Элвиса.