

Практическая работа 6. Условные обозначения в Kotlin

Прежде чем начать

- Условные выражения - одна из важнейших основ программирования. Условия - это команды в языках программирования, которые обрабатывают решения. С помощью условий код становится динамичным, что означает, что он может вести себя по-разному в зависимости от другого условия.

Эта практическая работа научит вас использовать операторы и выражения `if/else` и `when` для написания условий в Kotlin.

Необходимые условия

- Знание основ программирования на Kotlin, включая переменные, функции `println()` и `main()`.

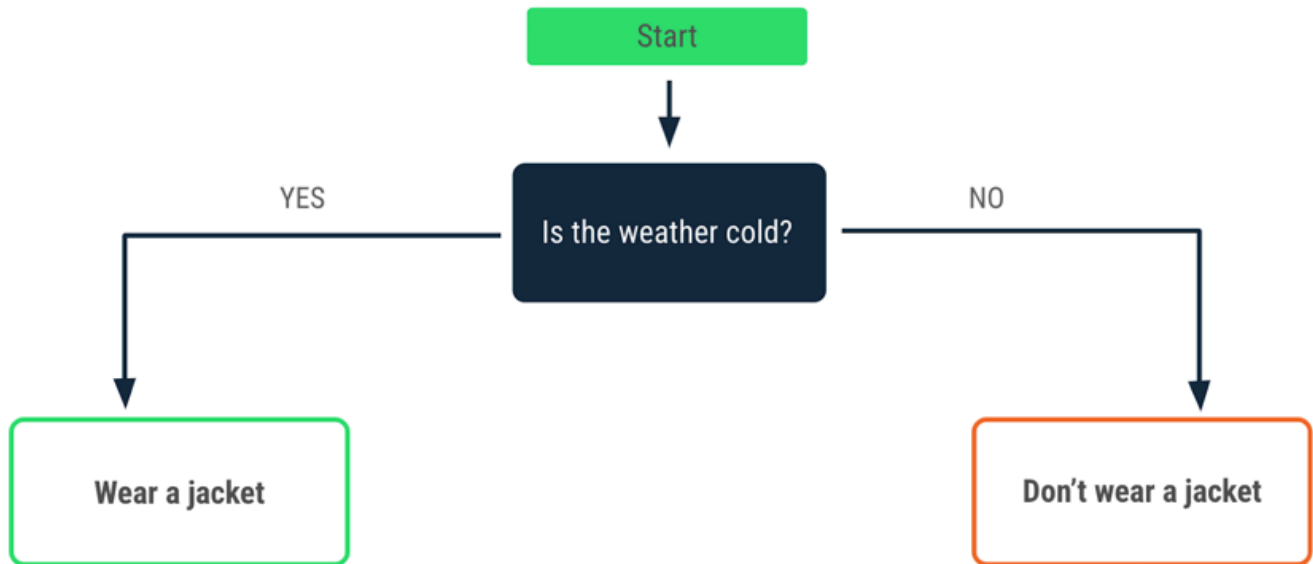
Что вы узнаете

- Как писать булевы выражения.
- Как писать операторы `if/else`.
- Как писать операторы `when`.
- Как использовать запятые, чтобы определить общее поведение для нескольких ветвей в условии `when`.
- Как использовать диапазон `in`, чтобы определить общее поведение для диапазона ветвей в условиях `when`.
- Как использовать ключевое слово `is` для написания условных выражений `when`.

Что вам понадобится IDE IntelliJ Idea

Используйте операторы `if/else` для выражения условий

В жизни часто приходится поступать по-разному в зависимости от ситуации, в которой вы оказались. Например, если погода холодная, вы надеваете куртку, а если теплая, то не надеваете.



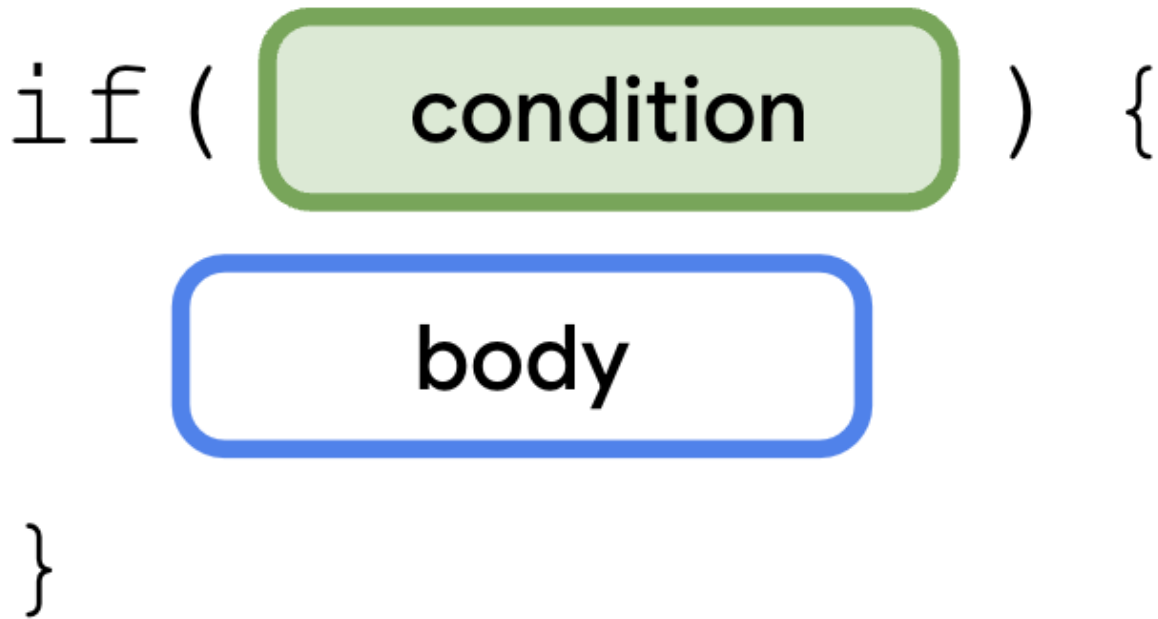
Принятие решений также является фундаментальной концепцией в программировании. Вы пишете инструкции о том, как программа должна вести себя в той или иной ситуации, чтобы она могла действовать или реагировать соответствующим образом, когда ситуация возникнет. В Kotlin, когда вы хотите, чтобы ваша программа выполняла различные действия в зависимости от условия, вы можете использовать оператор `if/else`. В следующем разделе вы напишете оператор `if`.

Запись условий `if` с помощью булевых выражений

Представьте, что вы создаете программу, которая сообщает водителям, что им следует делать, когда они стоят на светофоре. Сосредоточьтесь на первом условии: красный сигнал светофора. Что вы делаете на красный сигнал светофора? Остановиться!



В Kotlin это условие можно выразить с помощью оператора `if`. Посмотрите на анатомию оператора `if`:



Чтобы использовать операторы `if`, необходимо использовать ключевое слово `if`, за которым следует условие, которое вы хотите оценить. Условие должно быть выражено булевым выражением. Выражения объединяют значения, переменные и операторы, которые возвращают значение. Булевы выражения возвращают булево значение.

Ранее вы узнали об операторах присваивания, таких как:

```
val number = 1
```

Оператор присваивания = присваивает числовой переменной значение 1.

В отличие от этого, булевы выражения строятся с помощью операторов сравнения, которые сравнивают значения или переменные с обеих сторон уравнения. Рассмотрим один из операторов сравнения.

```
1 == 1
```

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

Найдите булево значение этого выражения:

Используйте **Idea** для запуска вашего кода.

В теле функции добавьте функцию `println()`, а затем передайте ей в качестве аргумента выражение `1 == 1`:

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

Запустите программу, а затем посмотрите ее результаты:

```
true
```

Первое значение 1 равно второму значению 1, поэтому булево выражение возвращает значение true, которое является булевым значением.

Попробуйте Помимо оператора сравнения ==, существуют дополнительные операторы сравнения, которые можно использовать для создания булевых выражений:

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

Отработайте использование операторов сравнения с простыми выражениями:

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

```
fun main() {  
    println(1 < 1)  
}
```

Запустите программу, а затем посмотрите вывод: Вывод возвращает ложное значение, потому что первое значение 1 не меньше второго значения 1.

```
false
```

Повторите первые два шага с другими операторами сравнения и числами.

- Напишите простой оператор if Теперь, когда вы увидели несколько примеров написания булевых выражений, вы можете написать свой первый оператор if. Синтаксис оператора if выглядит следующим образом:

```
if ( condition ) {  
    body  
}
```

Оператор `if` начинается с ключевого слова `if`, за которым следует условие, представляющее собой булево выражение в круглых скобках и набор фигурных скобок. Тело - это ряд утверждений или выражений, которые вы помещаете внутри пары фигурных скобок после условия. Эти операторы или выражения выполняются только при выполнении условия. Другими словами, операторы в фигурных скобках выполняются только тогда, когда булево выражение в ветви `if` возвращает истинное значение.

Напишите оператор `if` для условия красного сигнала светофора:

Внутри функции `main()` создайте переменную `trafficLightColor` и присвойте ей значение «Red»:

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

Добавьте оператор `if` для условия красного сигнала светофора, а затем передайте ему выражение `trafficLightColor == «Red»`:

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

В теле оператора `if` добавьте функцию `println()`, а затем передайте ей аргумент «Stop»:

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

В теле оператора if добавьте функцию println(), а затем передайте ей аргумент «Stop»:

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

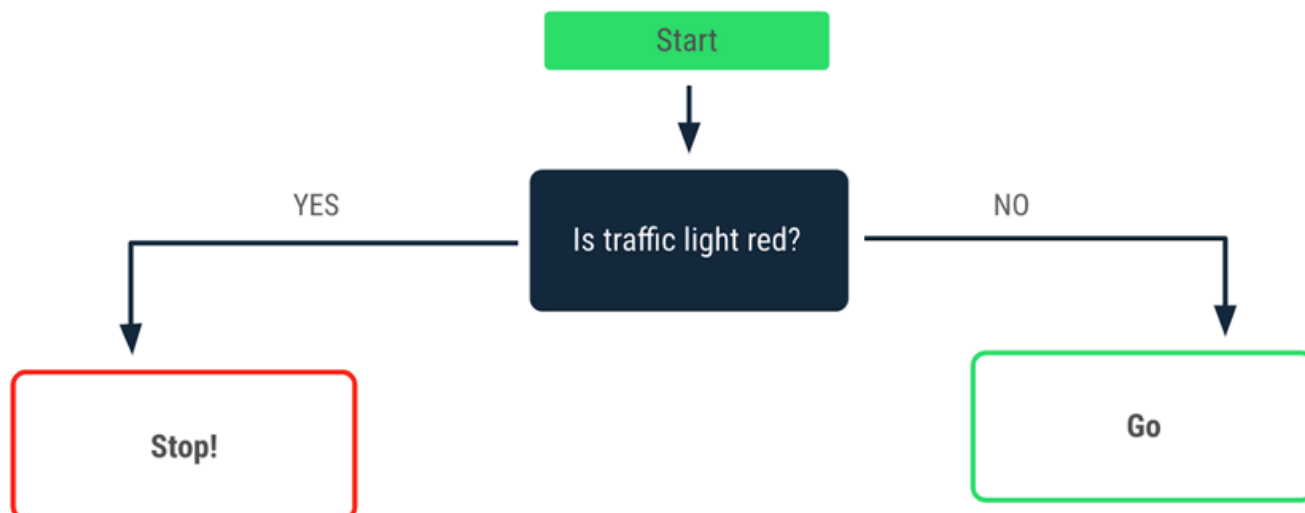
Запустите программу, а затем посмотрите ее результаты:

```
Stop
```

Выражение trafficLightColor == «Red» возвращает значение true, поэтому выполняется оператор println(«Stop»), который печатает сообщение Stop.

```
if (trafficLightColor == "Red") {  
    println("Stop")  
}
```

- Добавьте ветку else Теперь вы можете расширить программу, чтобы она указывала водителям, куда ехать, когда светофор не красный.



Чтобы создать оператор if/else, нужно добавить ветку else. Ветвь - это неполная часть кода, которую можно соединить, чтобы сформировать утверждения или выражения. Ветка else должна следовать за веткой if.

```
if ( condition ) {  
    body 1  
} else {  
    body 2  
}
```

После закрывающей фигурной скобки оператора if добавляется ключевое слово else, за которым следует пара фигурных скобок. Внутри фигурных скобок оператора else можно добавить второе тело, которое будет выполняться только в том случае, если условие в ветке if ложно.

Добавьте ветку else в свою программу:

После закрывающей фигурной скобки оператора if добавьте ключевое слово else, за которым следует еще одна пара фигурных скобок:

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

```
}  
}
```

Внутри фигурных скобок ключевого слова `else` добавьте функцию `println()`, а затем передайте ей аргумент «Go»:

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

Запустите эту программу, а затем просмотрите ее результаты:

```
Stop
```

Программа по-прежнему ведет себя так же, как и до добавления ветки `else`, но не печатает сообщение `Go`.

Переназначьте переменную `trafficLightColor` на значение «Green», потому что вы хотите, чтобы водители ехали на зеленый:

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

Запустите эту программу, а затем просмотрите ее результаты:

```
Go
```

Как видите, теперь программа печатает сообщение `Go` вместо сообщения `Stop`.


```
      boolean expression
      ↓
if (trafficLightColor == "Red") {
    println("Stop")
} else {
    println("Go")
}
```

← execute if true

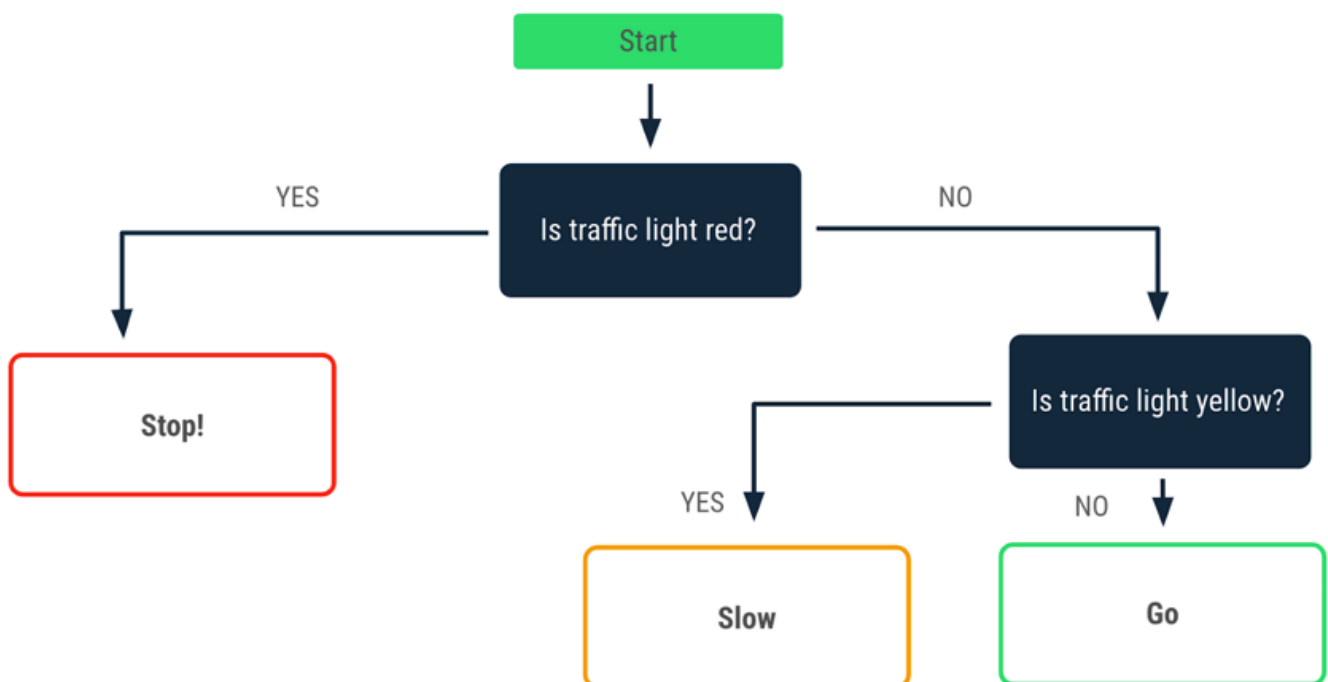
← execute if false

Вы переназначили переменную `trafficLightColor` на значение «Green», поэтому выражение `trafficLightColor == «Red»`, оцениваемое в ветви `if`, возвращает значение `false`, поскольку значение «Green» не равно значению «Red».

В результате программа пропускает все утверждения в ветви `if` и вместо этого выполняет все утверждения в ветви `else`. Это означает, что функция `println(«Go»)` выполняется, но функция `println(«Stop»)` не выполняется.

Добавьте ветку `else if`.

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



Вы научились писать условные сигналы, обслуживающие одну точку принятия решения, с помощью операторов `if/else`, содержащих одну ветвь `if` и одну ветвь `else`. Как справиться с более сложными

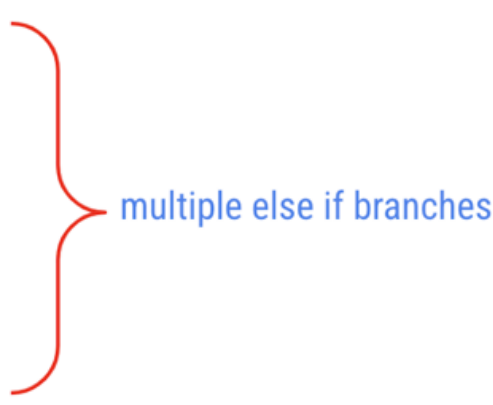
ветвлениями с несколькими точками принятия решений? Когда вы сталкиваетесь с несколькими точками принятия решений, вам нужно создавать условия с несколькими уровнями условий, что можно сделать, добавив ветви `else if` в операторы `if/else`.

После закрывающей фигурной скобки ветви `if` нужно добавить ключевое слово `else if`. Внутри круглых скобок ключевого слова `else if` нужно добавить булево выражение в качестве условия для ветви `else if`, а затем тело внутри пары фигурных скобок. Тело будет выполнено только в том случае, если условие 1 не выполняется, но выполняется условие 2.

```
if ( condition 1 ) {  
    body 1  
} else if ( condition 2 ) {  
    body 2  
} else {  
    body 3  
}
```

Ветвь `else if` всегда располагается после ветви `if`, но перед ветвью `else`. Вы можете использовать несколько ветвей `else if` в одном операторе:

```
if ( condition 1 ) {  
    body 1  
} else if ( condition 2 ) {  
    body 2  
} else if ( condition 3 ) {  
    body 3  
} else {  
    body 4  
}
```



multiple else if branches

Оператор if также может содержать ветвь if и ветвь else if без ветви else:

```
if ( condition 1 ) {  
    body 1  
} else if ( condition 2 ) {  
    body 2  
}
```

- Добавьте в программу ветвь else if:

После закрывающей фигурной скобки оператора if добавьте выражение else if (trafficLightColor == «Yellow»), за которым следуют фигурные скобки:

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

Внутри фигурных скобок ветви else if добавьте оператор println(), а затем передайте ему строковый аргумент «Slow»:

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

- Переназначьте переменную trafficLightColor на строковое значение «Yellow»:

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

- Запустите эту программу, а затем просмотрите ее результаты:

Slow

Теперь программа печатает сообщение Slow (Медленно), а не Stop (Стоп) или Go (Вперед).

```
1 if (trafficLightColor == "Red") {  
    println("Stop")           ← execute if (1) true  
2 } else if (trafficLightColor == "Yellow") {  
    println("Slow")          ← execute if (1) false and (2) true  
    } else {  
        println("Go")        ← execute if (1) and (2) false  
    }
```

Вот почему печатается только сообщение Slow, а не другие строки:

- Переменной `trafficLightColor` присвоено значение «Yellow».
- Значение «Yellow» не равно значению «Red», поэтому логическое выражение ветви `if` (обозначенное на рисунке как 1) возвращает значение `false`. Программа пропускает все операторы внутри ветви `if` и не печатает сообщение `Stop`. Поскольку ветвь `if` возвращает ложное значение, программа переходит к оценке булева выражения в ветви `else if`. Значение «Yellow» равно значению «Yellow», поэтому логическое выражение ветви `else if` (на рисунке обозначено как 2) возвращает истинное значение. Программа выполняет все операторы внутри ветви `else if` и печатает сообщение `Slow`. Поскольку булево выражение ветви `else if` возвращает истинное значение, программа пропускает остальные ветви. Таким образом, все операторы в ветви `else` не выполняются, и программа не печатает сообщение `Go`. Попробуйте Вы заметили, что текущая программа содержит ошибку?

В Главе 1 вы узнали о типе ошибки, называемой ошибкой компиляции, когда Kotlin не может скомпилировать код из-за синтаксической ошибки в вашем коде, и программа не может быть запущена. Здесь вы сталкиваетесь с другим типом ошибок, называемым логической ошибкой, при которой программа может выполняться, но не выдает выходных данных, как предполагалось.

Предположительно, вы хотите, чтобы водители ездили только тогда, когда на светофоре горит зеленый цвет. А что, если светофор сломался и выключился? Хотите ли вы, чтобы водитель поехал или получил предупреждение о том, что что-то не так?

К сожалению, в текущей программе, если цвет светофора любой другой, кроме красного или желтого, водителю все равно рекомендуется ехать.

Исправьте эту проблему:

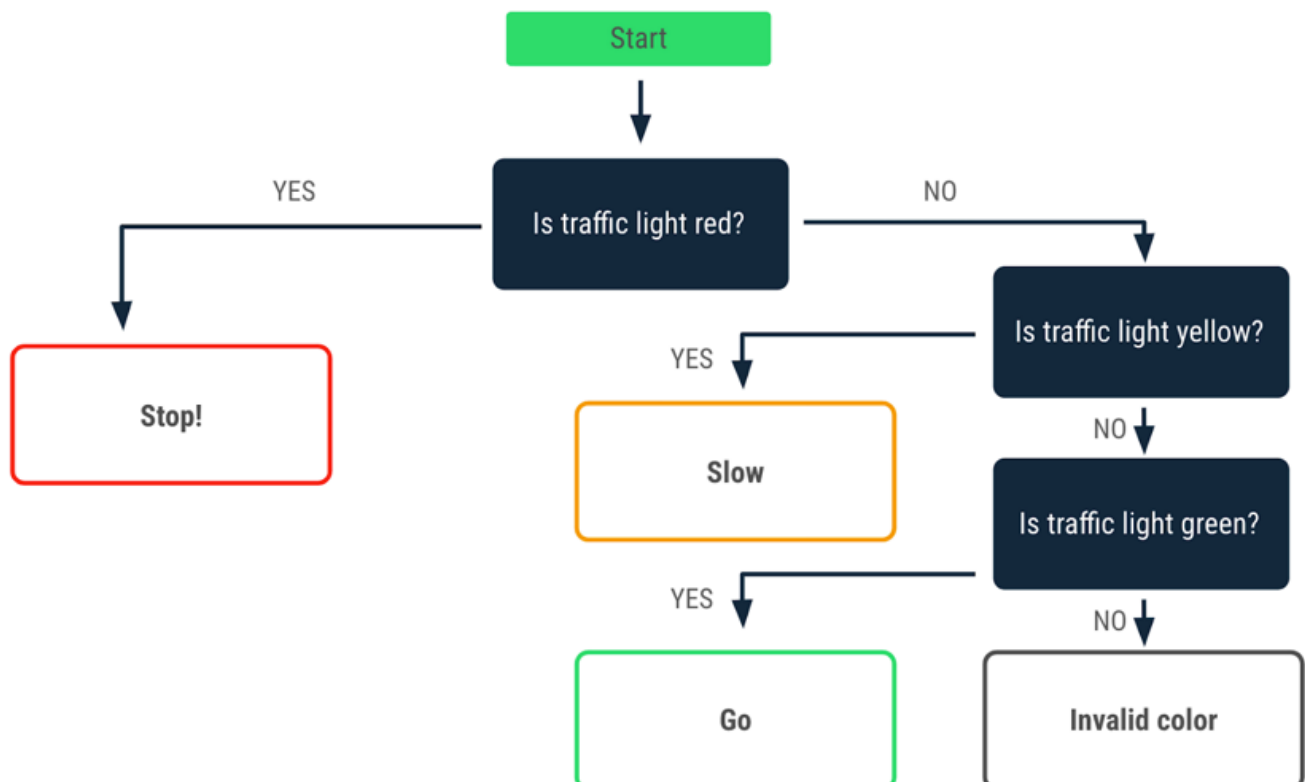
Переназначьте переменную `trafficLightColor` на значение «Black», чтобы проиллюстрировать выключенный светофор:

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

Запустите эту программу, а затем посмотрите ее результаты:

Go

Обратите внимание, что программа печатает сообщение Go, даже если переменной `trafficLightColor` не присвоено значение «Green». Можете ли вы исправить эту программу так, чтобы она отражала правильное поведение?



Необходимо изменить программу так, чтобы она печатала:

- Сообщение Go только в том случае, если переменной `trafficLightColor` присвоено значение «Green».

- Сообщение Invalid traffic-light color, если переменной trafficLightColor не присвоено значение «Red», «Yellow» или «Green».
- Исправьте ветвь else Ветвь else всегда располагается в конце оператора if/else, потому что это ветвь, позволяющая избежать ошибок. Она автоматически выполняется, когда все остальные условия в предыдущих ветвях не выполняются. Поэтому ветвь else не подходит для тех случаев, когда вы хотите, чтобы действие выполнялось только при выполнении определенного условия. В случае со светофором вы можете использовать ветвь else if, чтобы задать условие для зеленого света.
- Используйте ветвь else if для оценки условия зеленого сигнала светофора:

После текущей ветви else if добавьте еще одну ветвь else if (trafficLightColor == «Green»):

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

Запустите эту программу, а затем просмотрите вывод. Вывод пуст, потому что у вас нет ветви else, которая выполняется, когда предыдущие условия не выполняются.

После последней ветки else if добавьте ветку else с оператором println(«Неверный цвет светофора») 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")  
    }  
}
```

Запустите эту программу, а затем просмотрите ее результаты:

```
Invalid traffic-light color
```

Присвойте переменной `trafficLightColor` другое значение, кроме «Red», «Yellow» или «Green», а затем повторно запустите программу. Каков выход программы?

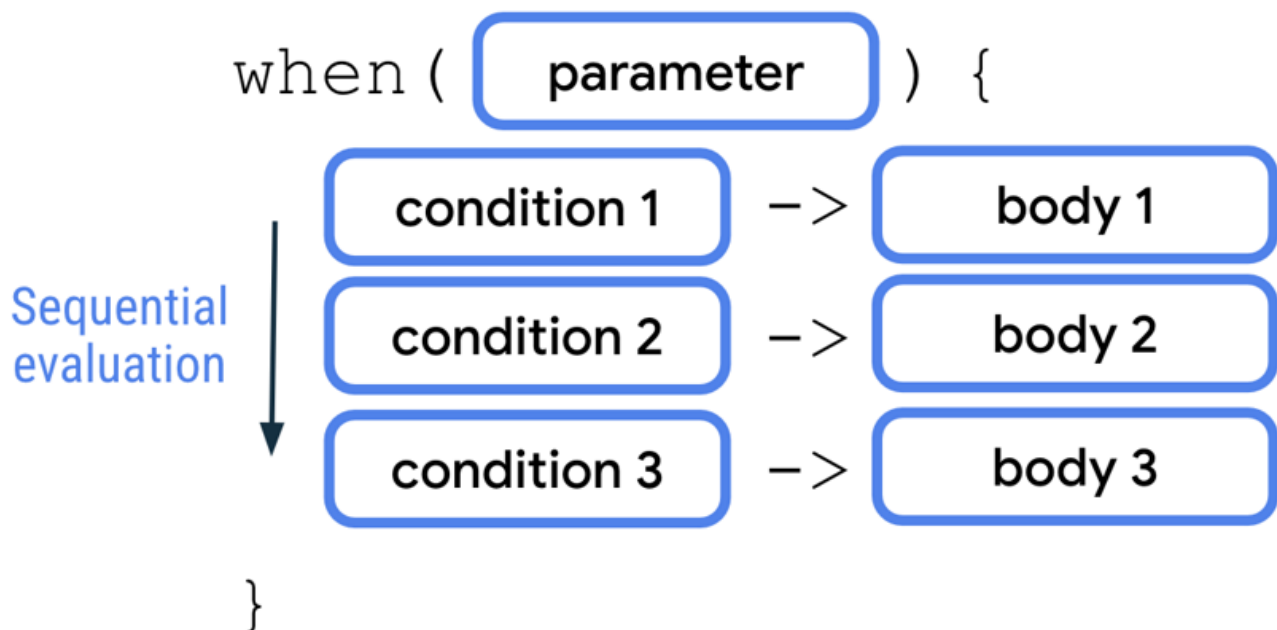
Хорошей практикой программирования является наличие явной ветви `else if` для проверки входных данных для зеленого цвета и ветви `else` для отлова других недействительных входных данных. Это гарантирует, что водители будут направлены на дорогу, только когда светофор зеленый. В остальных случаях передается явное сообщение о том, что светофор ведет себя не так, как ожидалось.

Используйте оператор `when` для нескольких ветвлений

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

В Kotlin при работе с несколькими ветвями можно использовать оператор `when` вместо оператора `if/else`, потому что это улучшает читаемость, то есть то, насколько легко читателям, обычно разработчикам, читать код. Очень важно учитывать удобочитаемость при написании кода, потому что, скорее всего, другим разработчикам придется просматривать и изменять ваш код на протяжении всего срока его службы. Хорошая читаемость гарантирует, что разработчики смогут правильно понять ваш код и не внесут в него случайно ошибки.

Операторы `when` предпочтительнее, когда необходимо рассмотреть более двух ветвей.



Оператор `when` принимает единственное значение через параметр. Затем это значение последовательно оценивается по каждому из условий. Затем выполняется соответствующее тело первого условия, которое было выполнено. Каждое условие и тело разделяются стрелкой (`->`). Как и в

операторах if/else, в операторах when каждая пара из условия и тела называется ветвью. Также, как и в операторе if/else, вы можете добавить ветку else в качестве заключительного условия в операторе when, которая будет работать в качестве промежуточной ветки.

Перепишите оператор if/else с помощью оператора when

В программе со светофором уже есть несколько ветвей:

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

Переделайте программу для использования оператора when:

В функции main() удалите оператор if/else:

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

Добавьте оператор when и передайте ему в качестве аргумента переменную trafficLightColor:

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

В тело оператора when добавьте условие «Red», за которым следует стрелка и тело println(«Stop»):

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

В следующей строке добавьте условие «Yellow», за которым следует стрелка и тело println(«Slow»):

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

```
}  
}
```

В следующей строке добавьте условие «Green», 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")  
    }  
}
```

В следующей строке добавьте ключевое слово `else`, за которым следует стрелка, а затем тело `println(«Недопустимый цвет светофора»)`:

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

Переназначьте переменную `trafficLightColor` на 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 результат будет получен при запуске этой программы?

Запустите программу, а затем просмотрите выходные данные:

Slow

```
when(trafficLightColor) {  
  1 "Red" -> println("Stop")  
  2 "Yellow" -> println("Slow")  
  3 "Green" -> println("Go")  
  4 else -> println("Invalid traffic-light color")  
}
```

На выходе получается сообщение Slow, потому что:

- Переменной trafficLightColor присвоено значение «Yellow».
- 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 оператор when.

Примечание: Существует вариант оператора when, который не принимает никаких параметров и используется вместо цепочки if/else.

Запись более сложных условий в операторе when

До сих пор вы узнали, как писать условия when для одного условия равенства, например, когда переменной trafficLightColor присваивается значение «Yellow». Далее вы научитесь использовать запятую (,), ключевое слово **in** и ключевое слово **is** для формирования более сложных условий when.

- Создайте программу, которая определяет, является ли число от 1 до 10 простым числом:
- Создайте консольный проект IntelliJ Idea в отдельном окне. К программе со светофором вы вернетесь позже.
- Определите переменную x и присвойте ей значение 3:

```
fun main() {  
    val x = 3  
}
```

- Добавьте оператор when, включающий несколько ветвей для условий 2, 3, 5 и 7, и сопровождайте каждую из них телом println(«x - простое число от 1 до 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.")  
    }  
}
```

- Добавьте ветку else с телом println(«x не является простым числом от 1 до 10.»):

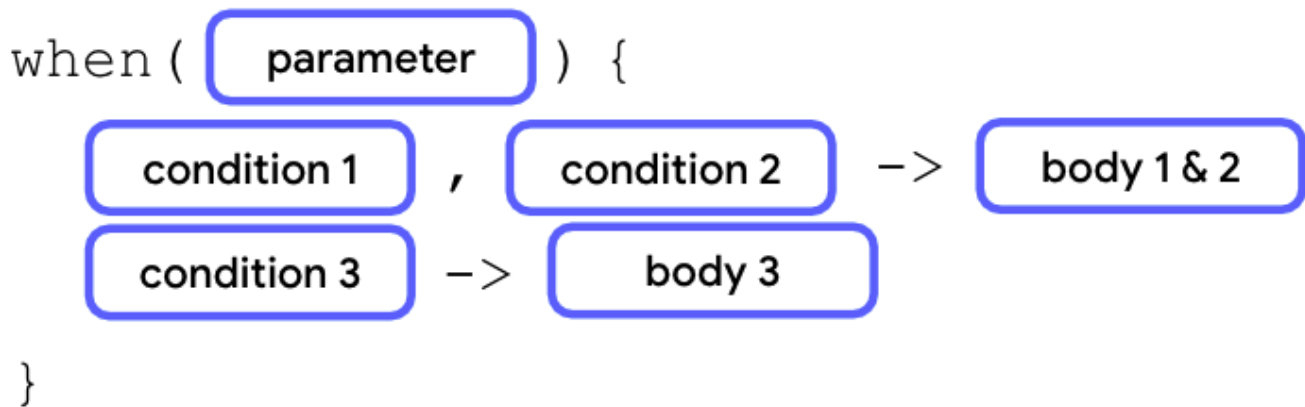
```
fun main() {  
    val x = 3  
  
    when (x) {  
        2 -> println("x is a prime number between 1 and 10.")  
        3 -> println("x is a prime number between 1 and 10.")  
        5 -> println("x is a prime number between 1 and 10.")  
        7 -> println("x is a prime number between 1 and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

- Запустите программу и убедитесь, что результат соответствует ожиданиям:

```
x is a prime number between 1 and 10.
```

- Используйте запятую (,) для нескольких условий
-

Программа prime-number содержит много повторений операторов println(). Когда вы пишете оператор when, вы можете использовать запятую (,) для обозначения нескольких условий, соответствующих одному и тому же телу.



В предыдущей схеме, если выполняется первое или второе условие, то выполняется соответствующее тело.

Перепишите программу для простых чисел с учетом этой концепции:

В ветвь для условия 2 добавьте 3, 5 и 7, разделенные запятыми (,):

```

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.")
    }
}

```

Удалите отдельные ветки для условий 3, 5 и 7:

```

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.")
    }
}

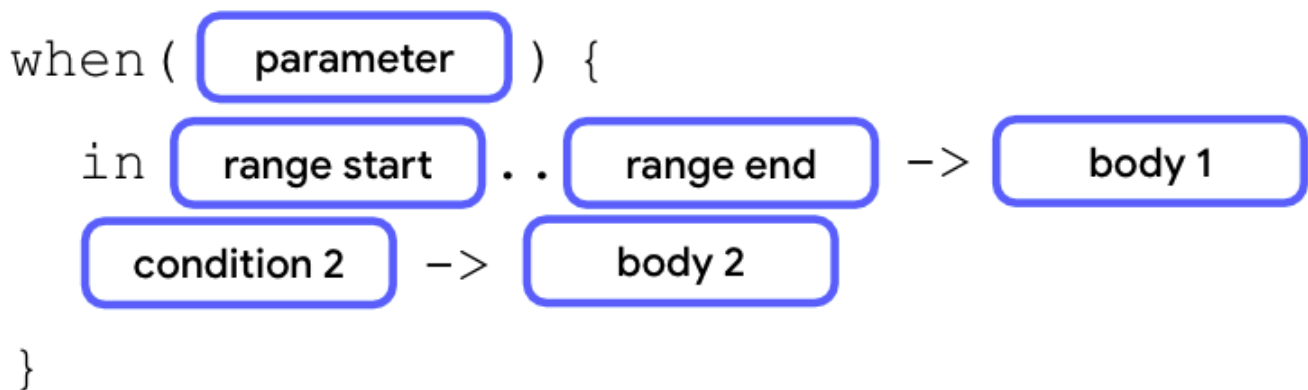
```

- Запустите программу и убедитесь, что результат соответствует ожиданиям:

x is a prime number between 1 and 10.

Использование ключевого слова **in** для диапазона условий

Помимо символа запятой (,) для обозначения нескольких условий, вы можете использовать ключевое слово `in` и диапазон значений в ветвях `when`.



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

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

Можно ли в программе для работы с простыми числами вывести сообщение, если число находится в диапазоне от 1 до 10, но не является простым?

- Добавьте еще одну ветвь с помощью ключевого слова `in`:

После первой ветви оператора `when` добавьте вторую ветвь с ключевым словом `in`, за которой следует диапазон `1..10` и тело `println`(«x - число от 1 до 10, но не простое»):

```

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.")
    }
}

```

- Измените переменную `x` на значение 4:

```

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.")
    }
}

```

```

    number.")
    else -> println("x isn't a prime number between 1 and 10.")
  }
}

```

- Запустите программу и проверьте вывод:

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

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

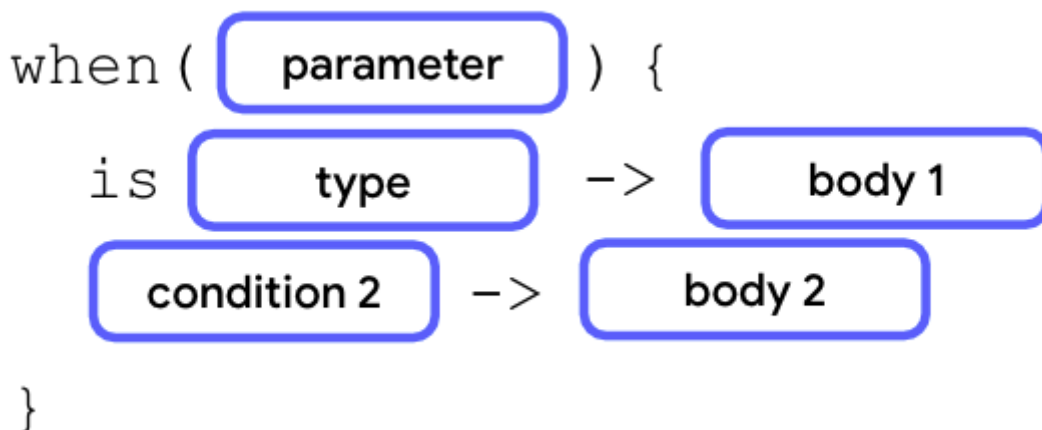
```

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

```

Вот как работает эта программа:

- Переменной `x` присваивается значение 4.
- Программа переходит к оценке условий для первой ветви. Значение 4 не является значением 2, 3, 5 или 7, поэтому программа пропускает выполнение тела первой ветви и переходит ко второй ветви.
- Значение 4 находится между 1 и 10, поэтому сообщение `x` - это число между 1 и 10, но не простое число. тело печатается. Выполняется одно тело, поэтому программа переходит к выходу из оператора `when` и игнорирует ветку `else`. Использование ключевого слова `is` для проверки типа данных Вы можете использовать ключевое слово `is` в качестве условия для проверки типа данных оцениваемого значения.



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

Можете ли вы вывести сообщение в своей программе для работы с простыми числами, если на вход подается целое число, выходящее за пределы диапазона от 1 до 10?

Добавьте еще одну ветвь с ключевым словом `is`:

- Измените x так, чтобы оно имело тип Any. Это указывает на то, что x может иметь значение, отличное от типа Int.

```
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.")  
    }  
}
```

- После второй ветви оператора when добавьте ключевое слово is и тип данных Int с телом println(«x - целое число, но не от 1 до 10»):

```
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.")  
    }  
}
```

- В ветке else измените тело на println(«x не является целым числом.»):

```
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.")  
    }  
}
```

- Измените переменную x на значение 20:


```
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.")  
    }  
}
```

- Запустите программу и проверьте вывод:

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

Программа выводит сообщение третьей ветви, но не сообщения первой, второй или четвертой ветвей.

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

Вот как работает программа:

- Переменной `x` присваивается значение 20. Программа переходит к оценке условий для первой ветви. Значение 20 не является значением 2, 3, 5 или 7, поэтому программа пропускает выполнение тела первой ветви и переходит ко второй ветви.
- Значение 20 не находится в диапазоне от 1 до 10, поэтому программа пропускает выполнение тела второй ветви и переходит к третьей ветви.
- Значение 20 имеет тип `Int`, поэтому `x` является целым числом, но тело выводится не в диапазоне от 1 до 10.
- Выполняется одно тело, поэтому программа переходит к выходу из оператора `when` и игнорирует ветку `else`.

Попробуйте Теперь отработайте полученные знания в программе со светофором.

Представьте, что в некоторых странах существует янтарный цвет светофора, который предупреждает водителей так же, как и желтый в других странах. Можете ли вы изменить программу так, чтобы это дополнительное условие было учтено, сохранив при этом исходные условия?

- Добавьте дополнительное условие к программе «Светофор»:

```
fun main() {  
    val trafficLightColor = "Yellow"
```

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

Во второй ветке оператора when добавьте запятую после условия «Желтый», а затем условие «Янтарный»:

```
fun main() {
    val trafficLightColor = "Yellow"

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

Измените переменную trafficLightColor на значение «Amber»:

```
fun main() {
    val trafficLightColor = "Amber"

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

Запустите эту программу и проверьте результат:

Slow

Использование if/else и when в качестве выражений

Вы узнали, как использовать if/else и when в качестве выражений. Когда вы используете условия как утверждения, вы позволяете каждой ветви выполнять различные действия в теле в зависимости от условий.

Вы также можете использовать условия в качестве выражений, чтобы возвращать различные значения для каждой ветви условия. Если тело каждой ветви выглядит одинаково, можно использовать условные выражения для улучшения читаемости кода по сравнению с условными операторами.

```
val name = if ( condition ) {  
    body 1  
} else {  
    body 2  
}
```

Синтаксис условных выражений похож на синтаксис операторов, но последняя строка тела в каждой ветке должна возвращать значение или выражение, а условные выражения присваиваются переменной.

Если тела содержат только возвращаемое значение или выражение, можно убрать фигурные скобки, чтобы сделать код более кратким.

```
val name = if ( condition ) expression 1 else expression 2
```

В следующем разделе вы рассмотрите выражения if/else на примере программы «Светофор».

Преобразование оператора if в выражение

В этом выражении if/else много повторений оператора println():

```
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 оператор if/else в выражение if/else и удалите повторение:
- Определите переменную message, а затем присвойте ей оператор if/else:

```
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")  
    }  
}
```

- Удалите все операторы println() и 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"  
}
```

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

```
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)  
}
```

- Запустите эту программу, а затем просмотрите ее результаты:

Invalid traffic-light color

Попробуйте Переделайте программу светофора так, чтобы она использовала выражение `when` вместо оператора `when`:

```
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 оператор `when` в выражение, чтобы не повторять операторы `println()`?

- Создайте переменную `message` 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"  
    }  
}
```

Добавьте оператор `println()` в последнюю строку программы, а затем передайте ему переменную `message` в 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: Выражение `when` не нуждается в определении ветви `else`. Однако в большинстве случаев выражение `when` требует ветви `else`, поскольку выражение `when` должно возвращать значение. Поэтому компилятор Kotlin проверяет, являются ли все ветви исчерпывающими. Ветвь `else` гарантирует, что не возникнет сценария, в котором переменной не будет присвоено значение.

Заключение

Вы узнали об условных выражениях и о том, как писать их в Kotlin.

Резюме

- В Kotlin ветвление может быть реализовано с помощью условий `if/else` или `when`.
- Тело ветви `if` в условии `if/else` выполняется только тогда, когда булево выражение внутри условия ветви `if` возвращает истинное значение.
- Последующие ветви `else if` в условии `if/else` выполняются только тогда, когда предыдущие ветви `if` или `else if` возвращают ложные значения.
- Последняя ветвь `else` в условии `if/else` выполняется только тогда, когда все предыдущие ветви `if` или `else if` возвращают ложные значения.
- Рекомендуется использовать условие `when`, чтобы заменить условие `if/else`, когда есть более двух ветвей.
- Вы можете писать более сложные условия в условиях `when` с помощью запятой (,), диапазонов и ключевого слова `is`.
- Условия `if/else` и `when` могут работать как утверждения или выражения.