Conditions with `when`

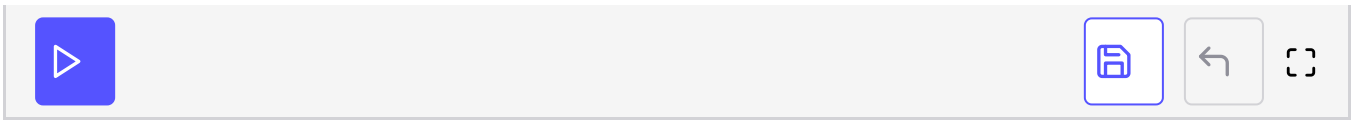
Understand Kotlin's `when` construct and best practices of using `if` vs `when`.

WE'LL COVER THE FOLLOWING

Conditions Using `when`

You can use `when` conditions to define different behaviors for a given set of distinct values. This is typically more concise than writing cascades of `if`-`else if` conditions. For instance, you can replace the `if` condition above with a `when` condition to save around 30% of the lines of code in this example:

```
when (planet) {
  "Jupiter" -> println("Radius of Jupiter is 69,911km")
  "Saturn"  -> println("Radius of Saturn is 58,232km")
  else      -> println("No data for planet $planet")
}
```

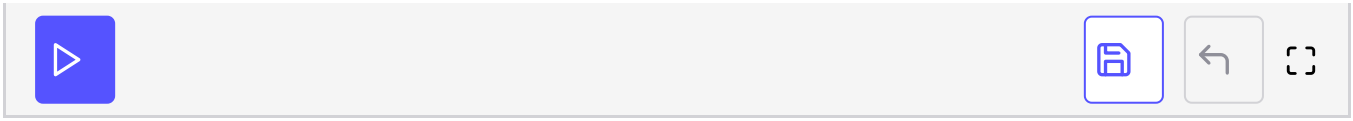


The `when` keyword is followed by the variable it compares against. Then, each line defines the value to check on the left-hand side, followed by an arrow `->` and the block of code to execute if the value matches.

The code block on the right-hand side must be wrapped in curly braces unless it's a single expression. Optionally, an `else`-block can be used to define a block of code to run if no other case matches.

This language construct is basically the same as `switch` from languages like C or Java. However, it's more powerful since it supports more complex checks:

```
when(temperatureInKelvin) {
      700              -> println("This is Mercury's max surface temperature")
      0, 1, 2          -> println("This i as cold as it can physically get")
      in 300..699      -> println("This temperature is possible on Mercury")
      !in 0..300       -> println("This is pretty hot")
      earthSurfaceTemp() -> println("This is earth's average surface temperature")
      is Int             -> println("The given temperature is of type Int")
      else               -> {
        // Example of a multiline code block
        println("Default case")
      }
    }
```



As you can see, there are various ways to define the value(s) to compare against:

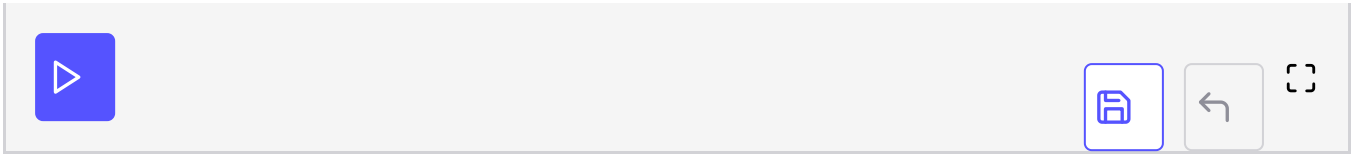
1. Fixed value (`700`)
2. Multiple fixed values (`0, 1, 2`)
3. Ranges (`in 300..699`)
4. Ranges with negation ("not in range") (`!in 0..300`)
5. Function call (`earthSurfaceTemp()`)
6. Type check (`is Int`)
7. Default case (`else`)

> **Note:** Kotlin adds a `break` to the end of each case implicitly so that you cannot introduce bugs due to a forgotten fallthrough case.

In fact, the `when` construct is even more powerful because it allows arbitrary conditions by leaving out the variable after the `when` keyword:

```
when {
      age >= 18 && !hasAccess -> println("Falsely rejected")
      age < 18 && hasAccess -> println("Falsely approved")
      else -> println("Correctly authorized")
```

```
}
```



Here, you can see the `age` variable is not mentioned in the `when` block's header as in `when (age)`. This way, you can use any boolean expressions on the left-hand sides of each `when` case, such as `age >= 18`.

Good Practice: `if` vs `when`

From a theoretical viewpoint, `if` and `when` are equally powerful. Coming from other languages, using `if` will feel more natural in most cases. Using `when` is definitely preferable when comparing against two or more distinct fixed values.

However, in aiming to write idiomatic Kotlin code, I'd encourage you to consider the `when` construct in more cases than the `switch` construct known from other languages. **In many cases, a `when` condition will be a more concise and readable alternative to a regular `if` condition**, especially whenever the right-hand sides are single expressions.

Quiz

Conditional control flow using `when`

1 Which left-hand side lets you check against a range of values inside a `when` condition?

Exercises

Rewrite all conditions from the previous lesson's exercise using `when`:

Complete the following code snippet to check whether...

1. The `age` variable is between 18 and 21 (both inclusive)
2. The `username` variable equals `"admin"` or `"system"`
3. The `number` variable is not equal to neither `17` nor `42`

...using only `when` conditions.

Problem | Solution

```
// Note: In this code widget, you have access to the predefined variables age, username, and

// Check age


// Check username


// Check number
```

Summary

Kotlin provides an `if` and a `when` construct for conditional control flow, which are equally expressive.

- `if` conditions are useful for simple if-then-else logic.
- `when` conditions are often more succinct if you have many branches, especially when checking against specific distinct values.

In the following lesson, you'll explore how to use `if` and `when` as expressions instead of statements.