When to Use

This lesson lists the various areas where std::string_view is recommended.

- Optimization: you can carefully review your code and replace various string operations with `string_view`. In most cases, you should end up with faster code and fewer memory allocations.
- As a possible replacement for const `std::string&` parameter - especially in functions that don't need the ownership and don't store the string.
- Handling strings coming from other API: *QString, CString, constchar**... everything that is placed in a contiguous memory chunk and has a basic char-type. You can write a function that accepts `string_view` and no conversion from that other implementation will happen.

In any case, it's important to remember that it's only a non-owning view, so if the original object is gone, the view becomes rubbish and you might get into trouble.

Moreover, `string_view` **might not contain null terminator** so your code has to support that as well.

For example, it's never a good idea to pass `string_view` to a function that accepts null-terminated strings. More on that in a separate section - about Risks with string_view.

We can see that `string_view` is a useful tool for performance. However, it is a sub-class of a more generic template class called `std::basic_string_view`. More on that in the next lesson.