Special case: optional<bool> and optional<T*>

This lesson highlights a case where using std::optional is not recommended.

While you can use optional on any type, you need to pay attention when trying to wrap boolean or pointers.

`optional<bool>` - what does it model? With such a construction, you have a tri-state bool. If you need it, then maybe it's better to look for a real tri-state bool like boost::tribool.

What's more, it might be confusing to use such type because `optional<bool>` converts to `bool`. Also, if there's a value inside then accessing that value returns `bool`.

Likewise, you have a similar ambiguity with pointers:

```
#include <iostream>
#include <optional>
using namespace std;

int main() {
  std::optional<int*> opi { new int(10) };
  if (opi && *opi)
  {
    std::cout << **opi << std::endl;
    delete *opi;
  }
  if (opi)
    std::cout << "opi is still not empty!";
}
```

Don't try doing it this way, it's just an example!

In the above example, you have to check `opi` to see if the optional is empty or not, but then the value of `opi` can also be `nullptr`.

The pointer to `int` is naturally "nullable", wrapping it into optional makes it

confusing to use.

Our discussion on `std::optional` has come to an end. Move to the next lesson for a summary of all that we have learned.