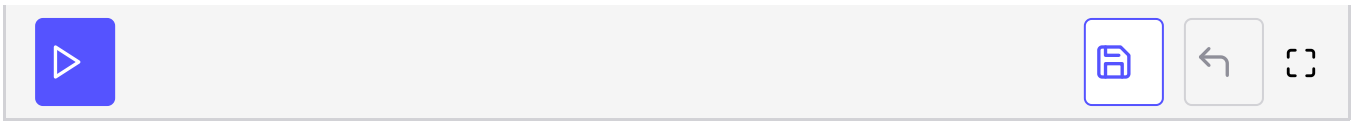
in_place Construction

Here, we will discuss why it is a good practice to use in_place with std::optional.

`std::optional` is a wrapper type, so you should be able to create optional objects almost in the same way as the wrapped object. And in most cases you can:

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

int main() {
  std::optional<std::string> ostr{"Hello World"};
  std::optional<int> oi{10};
  cout << *ostr << endl;
  cout << *oi << endl;
}
```



You can write the above code without stating the constructor such as:

```
std::optional<std::string> ostr{std::string{"Hello World"}};
std::optional<int> oi{int{10}};
```

Because `std::optional` has a constructor that takes U&& (r-value reference to a type that converts to the type stored in the optional). In our case it's recognized as `const char*` and strings can be initialized from it.

So what's the advantage of using `std::in_place_t` in std::optional?

There are at least two important reasons:

- Default constructor
- Efficient construction for constructors with many arguments

We'll build upon this in the next lesson, where we create `std::optional` with a default class constructor.