std::optional Creation

This lesson explains the different ways of creating std::optional variables.

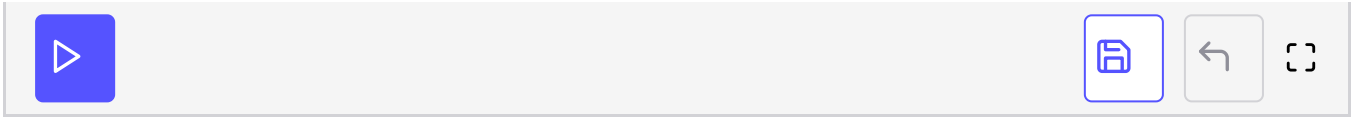
There are several ways to create std::optional:

- Initialise as empty
- Directly with a value
- With a value using deduction guides
- By using `make_optional`
- With `std::in_place`
- From other `optional`

See code below:

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

int main() {

  // empty:
  std::optional<int> oEmpty;
  std::optional<float> oFloat = std::nullopt;
  cout << "oEmpty = " << *oEmpty << endl;
  cout << "oFloat = " << *oFloat << endl;

  // direct:
  std::optional<int> oInt(10);
  std::optional oIntDeduced(10); // deduction guides
  cout << "oInt = " << *oInt << endl;
  cout << "oIntDeduced = " << *oIntDeduced << endl;

  // make_optional
  auto oDouble = std::make_optional(3.0);
  auto oComplex = std::make_optional<std::complex<double>>(3.0, 4.0);
  cout << "oDouble = " << *oDouble << endl;
  cout << "oComplex = " << *oComplex << endl;

  // in_place
  std::optional<std::complex<double>> o7{std::in_place, 3.0, 4.0};
  cout << "o7 = " << *o7 << endl;
```

```
  // will call vector with direct init of {1, 2, 3}
  std::optional<std::vector<int>> oVec(std::in_place, {1, 2, 3});


  // copy from other optional:
  auto oIntCopy = oInt;
  cout << "oIntCopy = " << *oIntCopy << endl;

}
```



As you can see in the above code sample, you have a lot of flexibility with the creation of optional. It's straightforward for primitive types, and this simplicity is extended even to complex types.

And, if you want the full control over the creation and efficiency, it's also good to know `in_place` helper types. We'll tackle that next.