std::optional

std::optional is very convenient when the value of our object can be null or empty.

std::optional is quite comfortable for calculations such as database queries that may have a result.

🔑 **Don't use no-results**
Before C++17 it was common practice to use a special value such as a null pointer, an empty string, or a unique integer to denote the absence of a result. These special values or no-results are very error-prone because you have to misuse the type system to check the return value. This means that for the type system that you have to use a regular value such as an empty string to define an irregular value.

The various constructors and the convenience function `std::make_optional` let you define an optional object `opt` with or without a value. `opt.emplace` will construct the contained value in-place and `opt.reset` will destroy the container value. You can explicitly ask a `std::optional` container if it has a value or you can check it in a logical expression. `opt.value` returns the value and `opt.value_or` returns the value or a default value. If `opt` has no contained value, the call `opt.value` will throw a `std::bad_optional_access` exception.

Here is a short example of using `std::optional`.

```
// optional.cpp
#include <iostream>
#include <optional>
#include <vector>

std::optional<int> getFirst(const std::vector<int>& vec){
  if (!vec.empty()) return std::optional<int>(vec[0]);
  else return std::optional<int>();
}

int main(){
```

```
int main(){

  std::vector<int> myVec{1, 2, 3};
  std::vector<int> myEmptyVec;

  auto myInt= getFirst(myVec);

  if (myInt){
    std::cout << *myInt << std::endl;                      // 1
    std::cout << myInt.value() << std::endl;               // 1
    std::cout << myInt.value_or(2017) << std::endl;        // 1
  }

  auto myEmptyInt= getFirst(myEmptyVec);

  if (!myEmptyInt){
    std::cout << myEmptyInt.value_or(2017) << std::endl;   // 2017
  }

  return 0;
}
```

std::optional

I use `std::optional` in the function `getFirst`. `getFirst` returns the first element if it exists. If not, you will get a `std::optional<int>` object. The main function has two vectors. Both invoke `getFirst` and return a `std::optional` object. In the case of `myInt` the object has a value; in the case of `myEmptyInt`, the object has no value. The program displays the value of `myInt` and `myEmptyInt`. `myInt.value_or(2017)` returns the value, but `myEmptyInt.value_or(2017)` returns the default value.

`std::variant`, explained in the next section, can have more than one value.