Automatic Return Type

In this lesson, we'll look at the technique that deduces return type automatically.

WE'LL COVER THE FOLLOWING

Automatic Return Type

A function template is automatically able to deduce their return type.

```
template <typename T1, typename T2>
auto add(T1 fir, T2 sec) -> decltype( fir + sec ) {
  return fir + sec;
}
```

> The automatic return type deduction is typically used for function templates but can also be applied to non-template functions.

Rules:

- `auto`: introduces the syntax for the delayed return type
- `auto`: `auto` type deduction is based on the function template argument deduction. Function template argument deduction (decays). So it means `auto` does not return the exact type but a decayed type such as for template argument deduction
- `decltype`: declares the return type
- The alternative function syntax is obligatory

> The C++11 syntax for automatically deducing the return type breaks the crucial principle of software development: DRY. DRY stands for **D**on't **R**epeat **Y**ourself.

Automatic Return Type: C++14

A function template is automatically able to deduce their return type.

```
template <typename T1, typename T2>
auto add(T1 fir, T2 sec){
    return fir + sec;
}
```

Rules

- **auto:** introduces the syntax for the delayed return type
- ~~`decltype`: declares the return type~~
- ~~The alternative function syntax is obligatory.~~

With the expression `decltype(auto)`, `auto` uses the same rules to determine the type as `decltype`. This means, in particular, no decay takes place.

- Both declarations are identical.

```
decltype(expr) v= expr;
decltype(auto) v= expr;
```

- The syntax also applies for the automatic return type of a function template.

```
template <typename T1, typename T2>
decltype(auto) add(T1 fir, T2 sec){
    return fir + sec;
}
```

> When a function template has more than one return statements, all return statements must have the same type.

In the next lesson, we'll study an example of automatic return type deduction.