Declaring Non-Type Template Parameters With auto

This lesson teaches you how C++ 17 uses auto for deducing non-type template parameters.

This is another part of the strategy to use `auto` everywhere. With `C++11` and `C++14` you can use it to deduce variables or even return types automatically, plus there are also generic lambdas. Now you can also use it for deducing non-type template parameters. For example:

```
template <auto value> void f() { }
...
f<10>(); // deduces int
```

This is useful, as you don't have to have a separate parameter for the type of non-type parameter. *Like in C++11/14:*

```
template <typename Type, Type value> constexpr Type TConstant = value;
...
constexpr auto const MySuperConst = TConstant<int, 100>;
```

With C++17 it's a bit simpler:

```
template <auto value> constexpr auto TConstant = value;
...
constexpr auto const MySuperConst = TConstant<100>;
```

No need to write `Type` explicitly.

As one of the advanced uses a lot of papers, and articles point to an example of heterogeneous compile time list:

```
template <auto ... vs> struct HeterogenousValueList {};
...
using MyList = HeterogenousValueList<'a', 100, 'b'>;
```

Before C++17 it was not possible to declare such list directly, some wrapper

class would have had to be provided first.

> **Extra Info:** The change was proposed in: P0127R2. In P0127R1, you can find some more examples and reasoning

---

The next lesson will conclude this section with the description of some other modifications in C++ 17.