Template Parameters

Let's familiarize ourselves with template parameters in this lesson.

WE'LL COVER THE FOLLOWING

Alias Templates

Alias templates aka template typedefs allow us to give a name to partially bound templates, which allows for partial specialization of templates.

```
template <typename T, int Line, int Col> class Matrix{
...
};
template <typename T, int Line>
using Square = Matrix<T, Line, Line>;
   template <typename T, int Line>
   using Vector = Matrix<T, Line, 1>;
```

Alias templates cannot be further specialized.

Template Parameters

Every template is parameterized by one or more template parameters, indicated in the parameter-list of the template.

C++ supports three different kinds of template parameters

1. **Type parameters**

```
std::vector<int> vec = {1, 2, 3, 4, 5};
```

```
std::vector<int> vec = {1, 2, 3, 4, 5};
```

2. **Non-type parameters**

```
std::array<int, 5> arr = {1, 2, 3, 4, 5};
```

3. **Template-template parameters**

```
template <typename T, template <typename, typename> class Cont> class Matr
ix{
...
Matrix<int, std::vector> myIntVec;
```

Types

A type parameter is a typical case for template arguments.

- Type parameters are class types and fundamental types

Non-Types

Non-types are template parameters which can be evaluated at compile-time.

The following types are possible:

- Integers and enumerations
- Pointers to objects, functions, and attributes of a class
- References to objects and functions
- `std::nullptr_t` constant

> With C++17, floating-point numbers and strings cannot be used as non-type parameters.

To learn more about template parameters, click here.

In the next lesson, we'll look at some examples of the three different types of template parameters.