# Templates

## Terms

| | |
|---|---|
| Class templates | Template argument |
| Function templates | Template declaration |
| Generics | Template parameter |

## Summary

- *Templates* allow us to create flexible functions and classes that can work with all data types. They're called *generics* in other programming languages like C#, Java, TypeScript, etc.

- To create a function/class template, we code a *template declaration* before the function/class. The declaration consists of the **`template`** keyword followed by a pair of angle brackets (<>).

- Within the angle brackets, we type one or more type parameters. The type of these parameters can be **`class`** or **`typename`**.

- By convention, we name these parameters **T**, **U**, **V**, but we can name them whatever that makes more sense.

- The compiler generates instances of a function/class template based on the usages. If we don't use a function/class template, it's not included in the executable.

- Methods of class templates should be implemented in the header file.

- When using templates, most of the time the compiler can guess the type of parameters. If not, we have to explicitly supply type arguments.

```cpp
// Function template
template<typename K, typename V>
void display(K key, V value) {}

// Class template
template<typename K, typename V>
class Pair {
public:
    Pair(K key, V value);
private:
    K key;
    V value;
};

template<typename K, typename V>
Pair<K, V>::Pair(K key, V value): key{key}, value{value} {
}
```