Class Templates

This lesson teaches how to use template types in classes in functions

We'll cover the following

Template Type Members

As another powerful feature of C++, you can also make *template classes*, which are *classes* that can have *members* of the **generic** type.

Example

For example:

```
template<class T>
class Score
{
private:
        T scorenumber;
public:
       Box(T args){scorenumber = args;}
};
```

This class we defined keeps score in the variable `scorenumber` which could be `int`, `float` or `double`.

This is how it's object will be declared:

```
Score<int> myscore(40);
// or
Score<double> myscore(23.9);
```

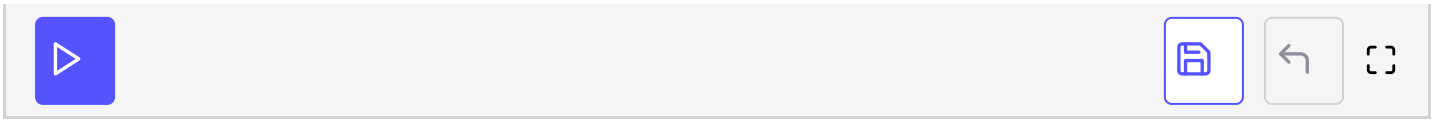
Let's take a look at another example where we have to find the **maximum** value by comparing **two** values.

```
#include <iostream>
using namespace std;

template <class T>
class myvalues {
    T myval1, myval2; //two values of type T
  public:
    myvalues (T arg1, T arg2){myval1=arg1; myval2=arg2;} //constructor
    T max (); //max function
};

template <class T>
T myvalues<T>::max()  //definition of a function with type T
{
  if(myval1 > myval2){
    return myval1;
  }else{
    return myval2;
  }
}

int main () {
  myvalues <int> obj(20, 50); //try changing the value and value types to results for different ty
  cout << "Max value is: " << obj.max();
  return 0;
}
```



As you can see in the example above this time we declared a function of **type** `T` in our class.

- The definition of the function was outside the class so we had to add the *prefix* for *template* class beforehand.

This marks the end of our discussion on *templates*. Next up we'll look at some more coding examples and will step through them in detail.