# Qt6 C++ Course

## What is Signal & Slots

In GUI programming, when we change one widget, we often want another widget to be notified.

More generally, we want objects of any kind to be able to communicate with one another.

For example, if a user clicks a Close button, we probably want the window's close() function

to be called.

Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to

a function, so if you want a processing function to notify you about some event you pass a

pointer to another function (the callback) to the processing function. The processing function

then calls the callback when appropriate. While successful frameworks using this method do exist,

callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness

of callback arguments. In Qt, we have an alternative to the callback technique: We use signals and slots.

Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks.

A signal is emitted when a particular event occurs. Qt's widgets have many predefined signals, but we can always subclass widgets to add our own signals to them. A slot is a function that is called in response to a particular signal. Qt's widgets have many pre-defined slots, but it is common practice to subclass widgets and add your own slots so that you can handle the signals that you are interested in. Signals and slots are made possible by Qt's meta-object system.

## What is Qt Meta Object System

Qt's meta-object system provides the signals and slots mechanism for inter-object communication, run-time type information, and the dynamic property system.

The meta-object system is based on three things:

The QObject class provides a base class for objects that can take advantage of the meta-object system.

The Q_OBJECT macro inside the private section of the class declaration is used to enable meta-object

features, such as dynamic properties, signals, and slots.

The Meta-Object Compiler (moc) supplies each QObject subclass with the necessary code to implement

meta-object features.