When to Use std::variant

This section explains some possible uses of std::variant

WE'LL COVER THE FOLLOWING

Unless you're doing some low-level stuff, possibly only with simple types, then unions might be a valid option [^noteGuide]. But for all other uses cases, where you need alternative types, `std::variant` is the way to go.

> **[^noteGuide]**: See C++ Core Guidelines - Unions for examples of a valid use cases for unions.

Some possible uses:

• All the places where you might get a few types for a single field: so things like parsing command lines, ini files, language parsers, etc.

• Expressing efficiently several possible outcomes of a computation: like finding roots of equations.

• Error handling - for example you can return `variant<Object, ErrorCode>`. If the value is available, then you `return` Object otherwise you assign some error code.

• Finite State Machines.

• Polymorphism without `vtables` and inheritance (thanks to the visitor pattern).

A Functional Background

It’s also worth mentioning that variant types (also called a tagged union, a discriminated union, or a sum type) come from the functional language world and Type Theory.

The next lesson further elaborates on the creation and initialization of `std::variant`.