Accessing Stored Value

This part discusses how to access stored values using helper functions.

WE'LL COVER THE FOLLOWING

Wrong Way to Access a Stored Value

From all of the examples you've seen so far, you might get an idea of how to access the value. But let's make a summary of this vital operation.

First of all, even if you know what the currently active type is you cannot do:

```
std::variant<int, float, std::string> intFloatString { "Hello" };
std::string s = intFloatString;

// error: conversion from</span>
// 'std::variant<int, float, std::string>'<>
// to non-scalar type 'std::string' requested<>
// std::string s = intFloatString;<>
```

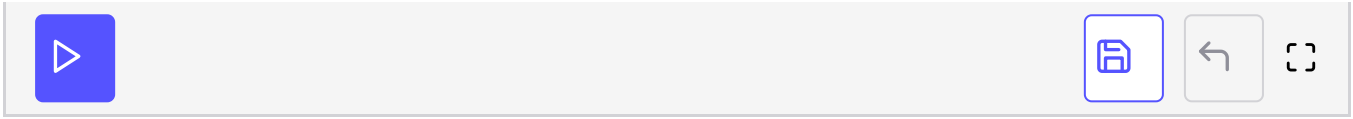
Using helper functions

You have `std::get<Type|Index>(variant)` which is a *non member* function. It returns a reference to the desired type if it's active (you can pass a Type or Index). If not then you'll get `std::bad_variant_access` exception.

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

int main() {
    std::variant<int, float, std::string> intFloatString;
    try
    {
        auto f = std::get<float>(intFloatString);
```

```
        std::cout << "float! " << f << '\n';
    }
    catch (std::bad_variant_access&)
    {
        std::cout << "our variant doesn't hold float at this moment...\n";
    }
}
```



The next option is `std::get_if`. This function is also a *non member* fucntion and won't throw. It returns a pointer to the active type or `nullptr`. While `std::get` needs a reference to the variant, `std::get_if` takes a pointer.

```
if (const auto intPtr = std::get_if<0>(&intFloatString))
std::cout << "int!" << *intPtr << '\n';
```

However, probably the most important way to access a value inside a variant is by using visitors.

The next lesson further elaborates on this.