# Accessing The Stored Value

Here are discussed the different modes of accessing a stored value in std::any.

To access the currently active value in `std::any` you have one option:

`std::any_cast<T>()`.

The function has three “modes” you can work with:

- read access - takes `std::any` as a reference, returns a copy of the value, throws `std::bad_any_cast` when it fails
- read/write access - takes `std::any` as a reference, returns a reference, throws `std::bad_any_cast` when it fails
- read/write access - takes `std::any` as a pointer, returns a pointer to the value (`const` or not) or `nullptr`

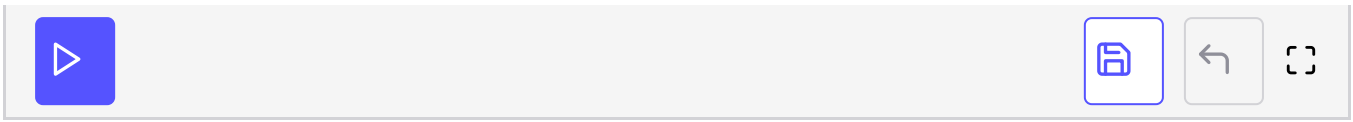
Concisely:

```
std::any var = 10;

// read access:
auto a = std::any_cast<int>(var);

// read/write access through a reference:
std::any_cast<int&>(var) = 11;

// read/write through a pointer:
int* ptr = std::any_cast<int>(&var);
*ptr = 12;
```

See the example:

```
#include <string>
#include <iostream>
#include <any>
#include <cassert>
#include <vector>
```

```
struct MyType
{

    int a, b;

    MyType(int x, int y) : a(x), b(y) { }

    void Print() {
        std::cout << a << ", " << b << '\n';
    }
};

int main()
{
    std::any var = std::make_any<MyType>(10, 10);
    try
    {
        std::any_cast<MyType&>(var).Print();
        std::any_cast<MyType&>(var).a = 11; // read/write
        std::any_cast<MyType&>(var).Print();
        std::any_cast<int>(var); // throw!
    }
    catch(const std::bad_any_cast& e)
    {
        std::cout << e.what() << '\n';
    }

    int* p = std::any_cast<int>(&var);
    std::cout << (p ? "contains int... \n" : "doesn't contain an int...\n");

    if (MyType* pt = std::any_cast<MyType>(&var); pt)
    {
        pt->a = 12;
        std::any_cast<MyType&>(var).Print();
    }
}
```



As you see, there are two options regarding error handling:

Via exceptions `(std::bad_any_cast)` or by returning a pointer (`or nullptr`).

The function overloads for `std::any_cast` pointer access are also marked with noexcept.

---

Let's consider some memory and performance issues with `std::any` in the next lesson.