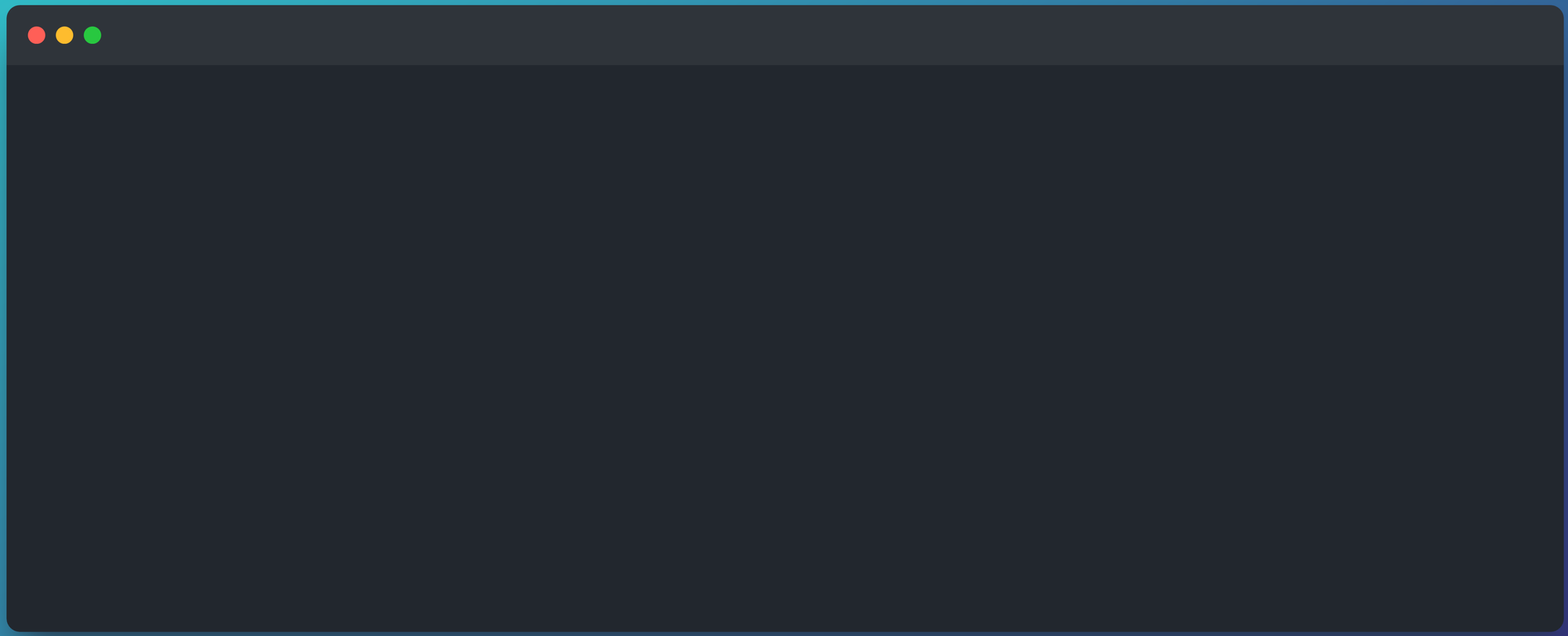
Exceptions: Miscellaneous topics.

```
/*
    . Exceptions - Miscellaneous topics:
        .#1: Rethrowing exceptions

        .#2: Custom termination

        .#3: Ellipsis catch all block

        .#4: Noexcept specifier

        .#5: Exceptions in destructors

*/
```

Exceptions: Rethrowing

```
for (size_t i{ 0 }; i < 5; ++i) {
    try {
        try {
            do_something(i);
        } catch (SomethingIsWrong &ex_inner) {
            if (typeid(ex_inner) == typeid(Warning)) {
                fmt::println(" {}-Inner catch block ,Exception caught: {}", typeid(ex_inner).name(), ex_inner.what());
            } else {
                throw;
                // throw ex_inner; // This will do a copy, and there will be slicing. Beware.
            }
        }
    } catch (SomethingIsWrong &ex_outer) {
        fmt::println(" {}-Outer catch block, Exception caught: {}", typeid(ex_outer).name(), ex_outer.what());
    }
} // End of for loop
```

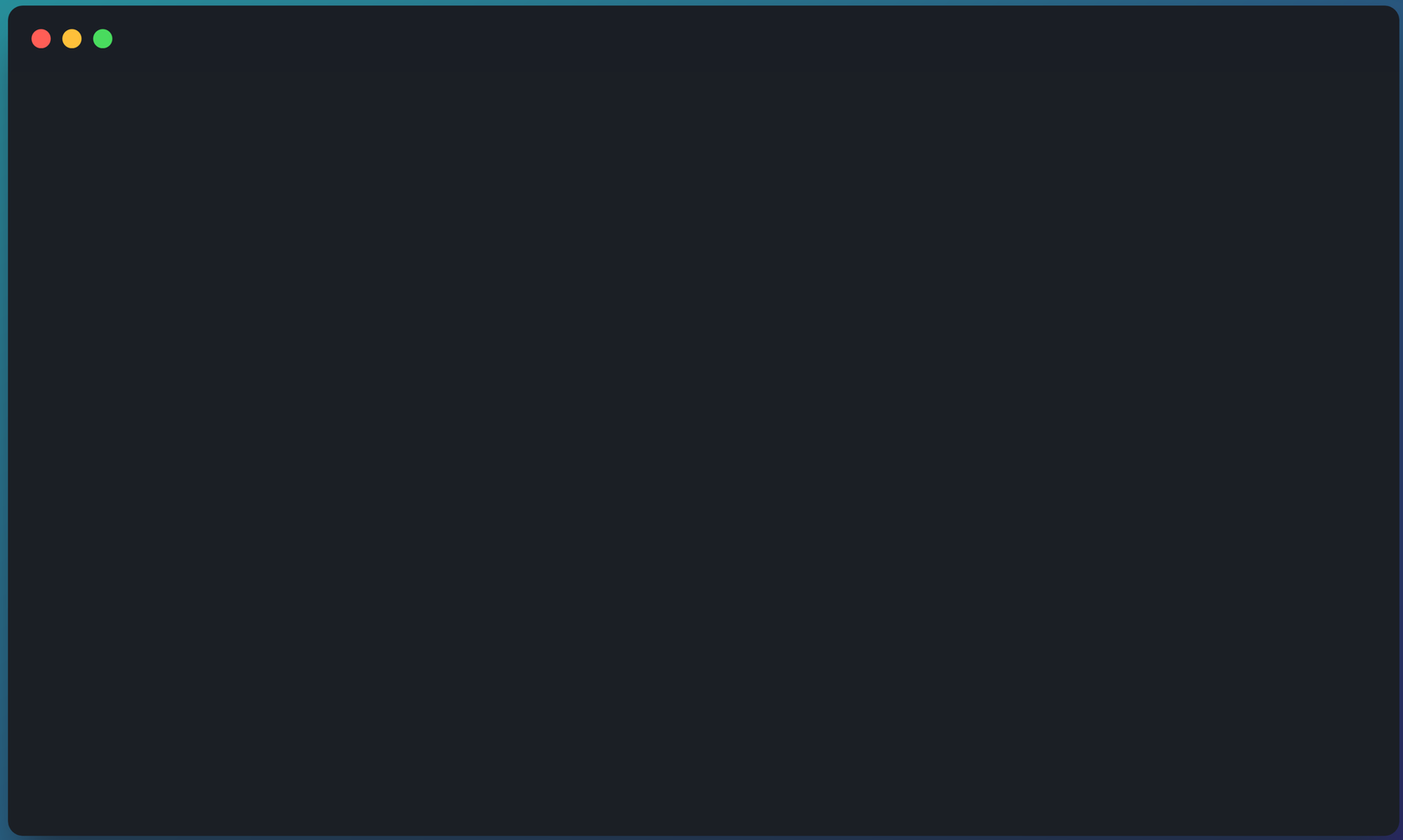
Exceptions: Rethrowing - Demo time!



Exceptions: Custom termination

```cpp
/*
    . Exploring custom termination:
        . C++ provides a way to set a custom termination function.
        . This function is called when an exception is thrown and not caught.
        . This is your chance to save what you can before the program terminates.
        . In our case we just print a message and wait 10s before calling std::abort().
        . std::abort is a function that terminates the program without unwinding the stack.
        . The code example for this lives in the main function.
*/
    void our_terminate_function()
    {
      fmt::println("Our custom terminate function called");
      fmt::println("Program will terminate in 10s ...");
      std::this_thread::sleep_for(std::chrono::milliseconds(10000));// Wait 10 more seconds
      std::abort();
    }

    int main() {

        // std::set_terminate(&our_terminate_function);
        std::set_terminate([]() {
            fmt::println("Our custom terminate function called");
            fmt::println("Program will terminate in 10s ...");
            std::this_thread::sleep_for(std::chrono::milliseconds(10000));// Wait 10 more seconds
            std::abort();
        });

        throw 1;
    }
```

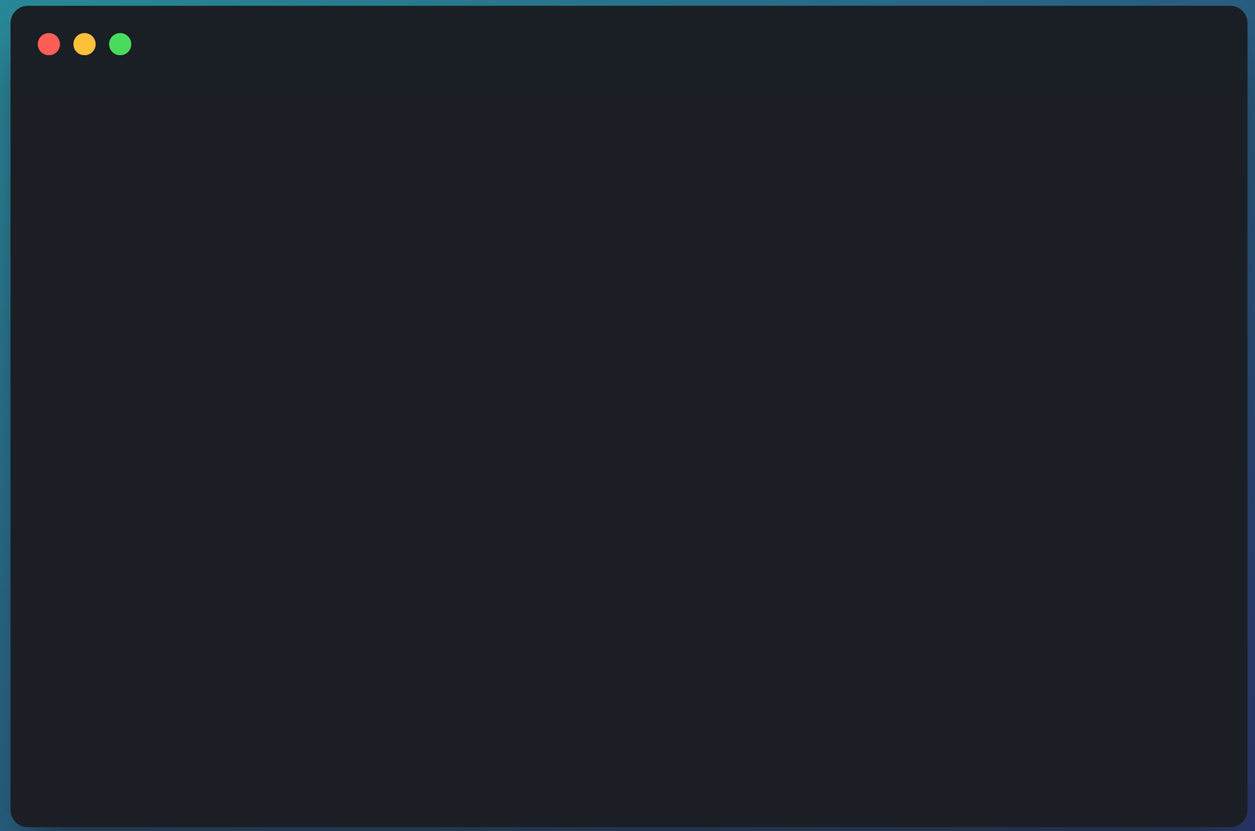
Exceptions: Custom termination - Demo time!



Exceptions: Ellipsis catch all block.

```cpp
//Ellipsis: A catch block that catches anything!
try {
  for (size_t i{}; i < 5; ++i) {
    try {
      some_function(i);
    } catch (int ex) {
      fmt::println("int handler - Caught an integer");
    } catch (...) {
      fmt::println("Inner... handler, Caught some exception");
      throw;
    }
  }
} catch (const std::string &ex) {
  fmt::println("Caught some string exception");
} catch (...) {
  fmt::println("Outer ...handler caught some other exception");
}
```

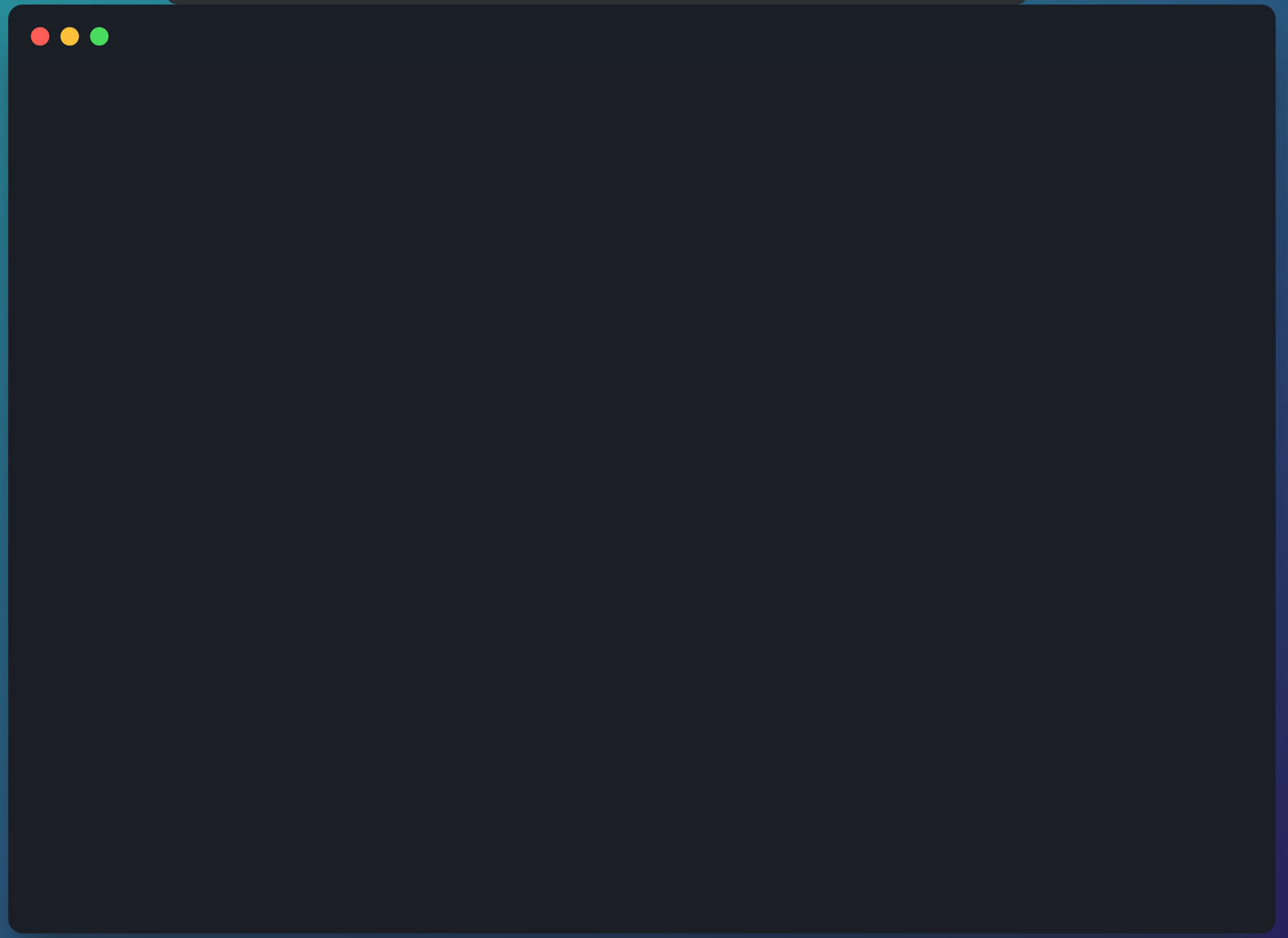
Exceptions: Ellipsis catch all block - Demo time!



Exceptions: noexcept

```
//The point: exceptions can't propagate out of the function
    export class Item {
    public:
        Item() {}

        void do_something_in_class() const noexcept {
            fmt::println("Doing something from class");
            try {
                throw 1;
            } catch (int ex) {
                fmt::println("Handling exception in Item::do_something_in_class");
                //throw; // Rethrowing in noexcept function/method will terminate program
            }
        }

    private:
        int m_member_var;
    };

    export void some_function() noexcept {
        try {
            throw 1;
        } catch (int ex) {
            fmt::println("Handling int exception in free function some_function()");
            //throw;
        }
    }
```

Exceptions: noexcept - Demo time!



Exceptions: destructors

```
/*
    . Exploring exceptions in destructors:
        . Exceptions in destructors are a bad idea.
        . Destructors are are noexcept by default.
        . Destructors are called as part of the stack unwinding process.
        . If an exception is thrown during stack unwinding, the program will terminate.

    . Note: You can allow exceptions to be thrown from destructors by using the noexcept(false) specifier.
        . This is not recommended. DON'T DO IT.
*/
```

Exceptions: destructors

```
export class Item
{
public:
    Item() {}
    //~Item() noexcept(false)
    //~Item() noexcept
    ~Item() // noexcept by default
    {
        //Uncomment the code below to show how bad it is to throw exceptions in destructors.
        /*
        try {
            throw 0;
        } catch (int ex) {
            throw;
        }
        */
    }
};
```

Exceptions: destructors - Demo time!