Iterator adaptors

```
/*
    . Exploring iterator adaptors:

        .#1: Common iterator adaptors:
            . They are:
                . back_insert_iterator
                . front_insert_iterator
                . insert_iterator
                . reverse_iterator

        .23: Stream iterators
            . std::istream_iterator and std::ostream_iterator are used to read from and write to streams.
*/
```

Iterator adaptors

```
/*
    . Some iterator adaptors:
        . C++ supports several iterator adapter types, each providing unique functionality
            to enhance the behavior of standard iterators.

        . Here is a list of the most common iterator adapters in C++:
            . std::back_insert_iterator
            . std::front_insert_iterator
            . std::insert_iterator
            . std::reverse_iterator
            . std::move_iterator
            . std::ostream_iterator
            . std::istream_iterator
            . std::istreambuf_iterator
            . std::ostreambuf_iterator
            . std::counting_iterator (C++20)
            . std::common_iterator (C++20)
            . std::unreachable_sentinel (C++20)
        . We cover the first five in this module.
*/
```

Iterator adaptors

```cpp
//std::back_inserter
// Initialize a vector with some elements. We use vector here because it supports push_back.
std::vector<int> vec = {1, 2, 3};
// Another vector with values to add to vec
std::vector<int> values_to_add = {4, 5, 6};

// Use std::copy to copy elements from values_to_add to vec
// std::back_inserter(vec) creates a back_insert_iterator for vec
std::copy(values_to_add.begin(), values_to_add.end(), std::back_inserter(vec));

// Output the results
for(int value : vec) {
    fmt::println("{}", value);
}
```

Iterator adaptors

```cpp
//std::front_inserter iterator
// Initialize a deque with some elements
// We use deque here because it supports push_front.
std::deque<int> deq = {3, 2, 1};
// A vector with values to add to the front of deq
std::vector<int> values_to_add = {4, 5, 6};

// Use std::copy to copy elements from values_to_add to the front of deq
// std::front_inserter(deq) creates a front_insert_iterator for deq
std::copy(values_to_add.begin(), values_to_add.end(), std::front_inserter(deq));

// Output the results
for(int value : deq) {
    fmt::println("{}", value);
}
```

Iterator adaptors

```
//std::inserter iterator: inserts at a specific location in the container
// Initialize a vector with some elements
std::vector<int> vec = {1, 2, 6, 7};
// A vector with values to insert into vec
std::vector<int> values_to_add = {3, 4, 5};

// Use std::copy to insert elements from values_to_add at the 3rd position in vec
// std::inserter(vec, vec.begin() + 2) creates an insert_iterator at the 3rd position
std::copy(values_to_add.begin(), values_to_add.end(), std::inserter(vec, vec.begin() + 2));

// Output the results
for(int value : vec) {
    fmt::println("{}", value);
}
```

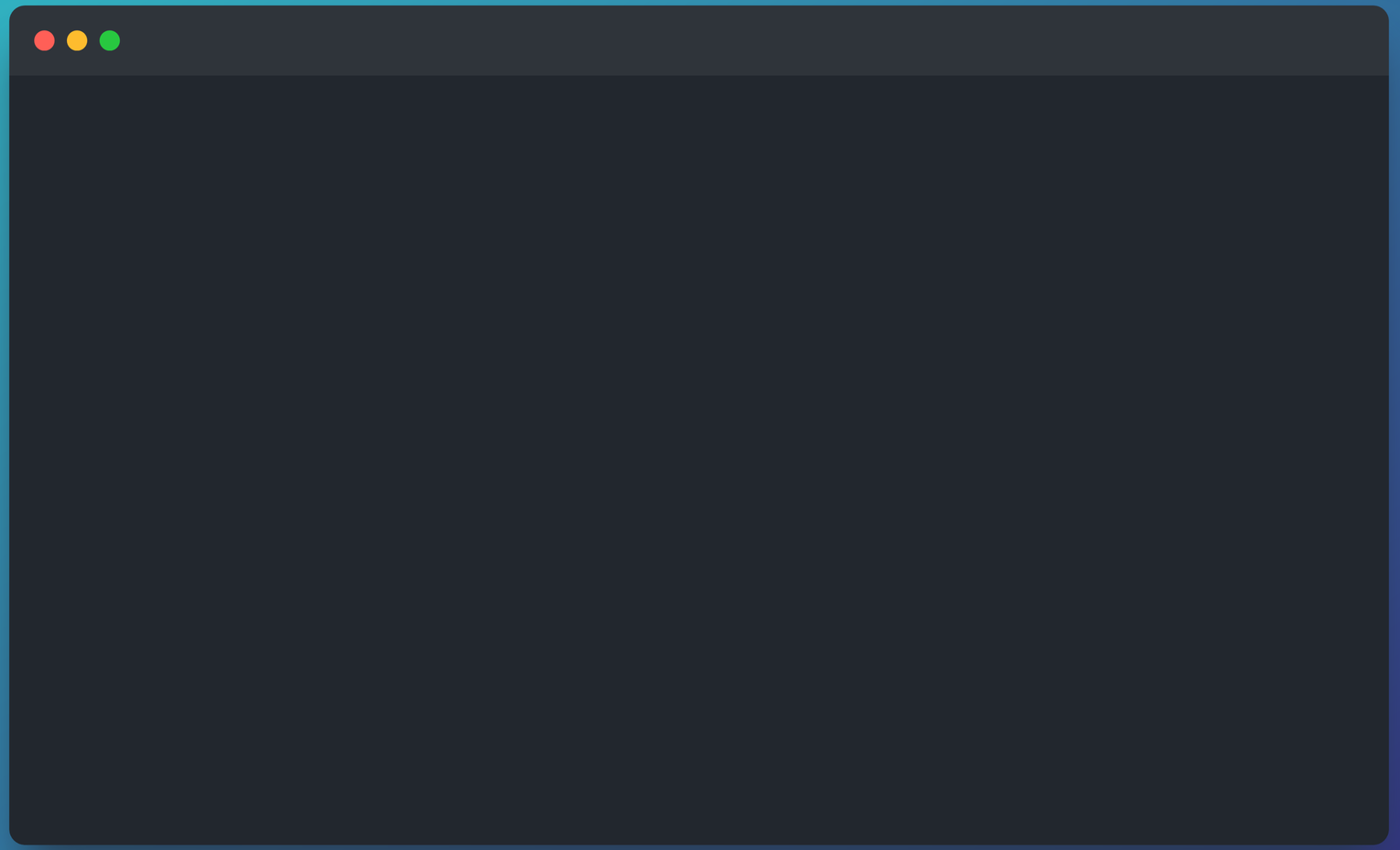
Iterator adaptors

```cpp
//reverse iterators are sometimes seen as adaptors
//Reverse iterators increment backwards.
// Initialize a vector with some elements
std::vector<int> vec = {1, 2, 3, 4, 5};

// Use std::for_each to print elements in reverse order
// vec.rbegin() and vec.rend() provide reverse iterators to the vector
// The lambda is a just a callback function in this case.
std::for_each(vec.rbegin(), vec.rend(), [](int value) {
    fmt::println("{}", value);
});

/*
    . Iteration and invocation:
        . std::for_each iterates of the range [vec.rbegin(), vec.rend()].
        . For each element, it invokes the lambda function.
        . The current element is passed to the lambda function as an argument.
*/
```

Demo time!



Stream iterators

```
/*
    . Exploring stream iteators
        . Using istream_iterator to read from std::cin
        . Using ostream_iterator to write to std::cout
        . These facilities work really well with the standard algorithms
*/
```

Stream iterators

```cpp
//Read data from input until the end of stream is reached.
//End of stream is reached when the user presses Ctrl+D on Unix or Ctrl+Z on Windows.
// Read integers from std::cin using istream_iterator
std::istream_iterator<int> inputIter(std::cin);
std::istream_iterator<int> endIter; // Default constructed end iterator

//Use a vector constructor that takes two iterators to initialize the vector
std::vector<int> numbers(inputIter, endIter); // Initialize vector with input

// Output the integers using ostream_iterator
std::ostream_iterator<int> outputIter(std::cout, " ");

std::cout << "You entered: ";
std::copy(numbers.begin(), numbers.end(), outputIter); // Copy elements to cout

std::cout << "\n";
```

Stream iterators

```cpp
//Some fun with stream iterators
//A function that sums up the elements in a range: [begin, end)
template <typename InputIt>
auto sum_up(InputIt begin, InputIt end){
    auto sum {*begin};
    for(auto iter = ++begin; iter != end; ++iter){
        sum += *iter;
    }
    return sum;
}


//Function to read fron cin, sum up and print the sum.
export void read_sum_print(){
    // Read integers from std::cin using istream_iterator
    std::istream_iterator<int> inputIter(std::cin);
    std::istream_iterator<int> endIter; // Default constructed end iterator

    auto result = sum_up(inputIter, endIter);
    std::cout << "The sum is: " << result << "\n";
}
```