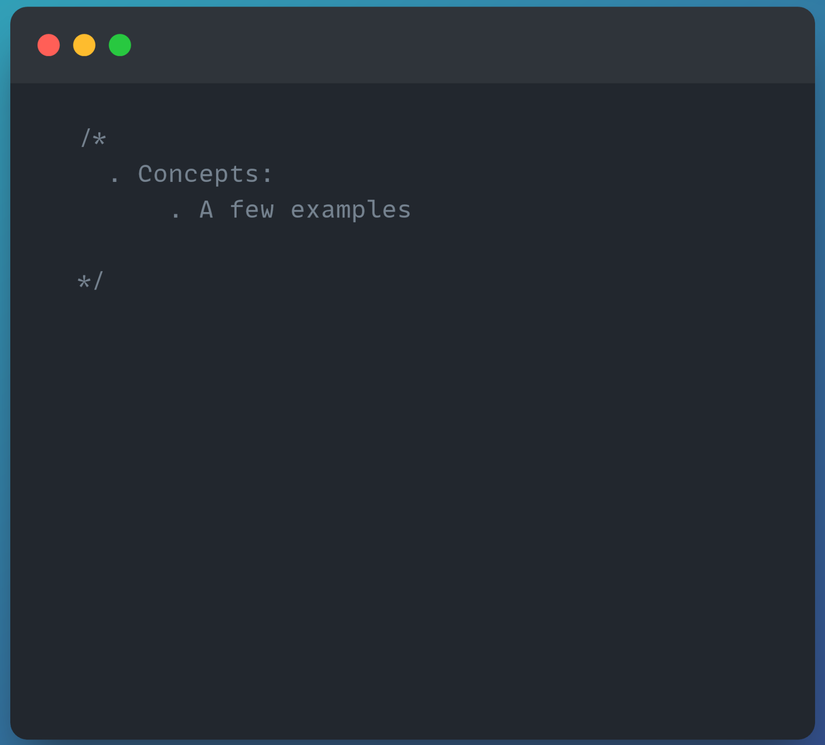
Concept examples



Concept examples

```
//Example #1
export template <typename T>
requires std::is_arithmetic_v<T>
class Point{
    /*
        static_assert(std::is_arithmetic_v<T>,
        "Coordinates of Point can only be numbers.");
    */
public :
    Point() = default;
    Point(T x, T y)
    : m_x(x), m_y(y)
    {
    }
    friend std::ostream& operator<< ( std::ostream& out, const Point<T> operand){
        out << "Point [ x : " << operand.m_x
                    << ", y : " << operand.m_y << "]";
        return out;
    }
private :
    T m_x;
    T m_y;
};
```

Concept examples

```
//Example #2: Built in concepts
//Numbers
//std::floating_point<T>
/*
static_assert(std::floating_point<int>);//Fails
static_assert(std::floating_point<double>);
*/

//equality and order
//https://en.cppreference.com/w/cpp/concepts/equality_comparable
//static_assert(std::equality_comparable<int>); // == , !=
//static_assert(std::equality_comparable_with<double,std::string>);

//static_assert(std::totally_ordered<int>);
//static_assert(std::totally_ordered<Point<int>>); // Needs all comparison operators
```

Concept examples

```
//Example #2: Built in concepts
 //Others
 //std::same_as
 //static_assert(std::same_as<int,int>); // Success
 //static_assert(std::same_as<int,double>);// Fail
 //static_assert(std::same_as<class_templates_concepts_02::Dog,class_templates_concepts_02::Cat>); //Fail
 static_assert(std::same_as<class_templates_concepts_02::Point<int>,class_templates_concepts_02::Point<int>>); // Success
 //static_assert(std::same_as<class_templates_concepts_02::Point<int>,class_templates_concepts_02::Point<float>>); // Fail


 //std::destructible
 //static_assert(std::destructible<Dog>);
 //std::derived_from : See official example  : https://en.cppreference.com/w/cpp/concepts/derived_from

 //std::copyable : needs a copy constructor
 //And much more
```

Concept examples

```cpp
//Example #3
// Define the OutputStreamable concept
export template <typename T>
concept OutputStreamable = requires(std::ostream& o, T d) {
    o << d;
};

// Define the Point class as a template
export template <OutputStreamable T>
struct Point {
    T mx; // Use template type T for x coordinate
    T my; // Use template type T for y coordinate

    // Friend function to overload the output operator for Point
    friend std::ostream& operator<<(std::ostream& o, const Point<T>& p) {
        o << "Point [ x : " << p.mx << ", y : " << p.my << "]";
        return o;
    }
};
```

Concept examples

```cpp
//Example #4
export template <typename T>
concept Number = (std::integral<T> || std::floating_point<T>)
                && !std::same_as<T, bool>
                && !std::same_as<T, char>;


export template <Number T, Number  U>
auto add ( T a, U b){
    return a + b;
}
```