Copy constructors

```
/*
      . Topics:
          . Copy semantics
          . Using the compiler genreated default constructor
          . Explicitly defaulting the copy constructor
          . Setting up our own copy constructor
          . Sometimes it's best not to set up a copy constructor AT ALL.
          . Copy constructors when dynamic memory is involved
          . When copy constructors are used


*/
```

```
ct8::Pixel p1(0x00FF00FF, 10, 20);
ct8::Pixel p2 = p1
```

Copy constructors

```
Pixel(const Pixel& other) = default;
Pixel(const Pixel& other);
```

```
//Copy constructor. No definition is needed if it's defaulted
Pixel::Pixel(const Pixel& other)
    : m_color{other.m_color},
    m_pos_x{other.m_pos_x},
    m_pos_y{other.m_pos_y}
{
    fmt::print("Pixel copied.\n");
}
```

Copy constructors: When dynamic memory is involved

```
/*
        . Compiler generated copy constructors do blind copies.
*/
export struct Position {
    unsigned int x{0};
    unsigned int y{0};
};

export class Pixel {
public:
    Pixel() = default;
    Pixel(uint32_t color, unsigned int x, unsigned int y);

    // Copy constructor for deep copy
    Pixel(const Pixel& other);

    ~Pixel();

    // Getters and setters for color
    uint32_t get_color() const;
    void set_color(uint32_t color);

private:
    uint32_t m_color{0xFF000000};
    Position* m_pos{nullptr}; // Raw pointer for position
    std::unique_ptr<Position> m_smart_pos; // Smart pointer for position
};
```

Copy constructors: When dynamic memory is involved

```cpp
// Constructor
Pixel::Pixel(uint32_t color, unsigned int x, unsigned int y)
    : m_color{color},
      m_pos{new Position{x, y}},
      m_smart_pos{std::make_unique<Position>(Position{x, y})}
{
}

// Copy constructor for deep copy with raw pointer and smart pointer
Pixel::Pixel(const Pixel& other)
    : m_color{other.m_color},
      m_pos{new Position{*other.m_pos}},
      m_smart_pos{std::make_unique<Position>(*other.m_smart_pos)} {
    fmt::print("Pixel deep copied (with both raw and smart pointers)\n");
}
```

Copy constructors: When they are used

```cpp
// Passing by value
void process_pixel(ct9::Pixel p) {
    fmt::print("Processing pixel: color={}, pos=({}, {})\n", p.get_color(), p.get_x(), p.get_y());
}

// Return by value
ct9::Pixel create_pixel() {
    ct9::Pixel p{0xFF0000, 30, 40};
    return p;  // Copy constructor may be invoked here (but likely optimized out)
}

// Copying an object in a collection (vector)
std::vector<ct9::Pixel> pixels;
pixels.push_back(p1);  // Copy constructor is called when adding to the vector

// Explicit copy constructor
ct9::Pixel p5(p1);  // Copy constructor is explicitly called
```