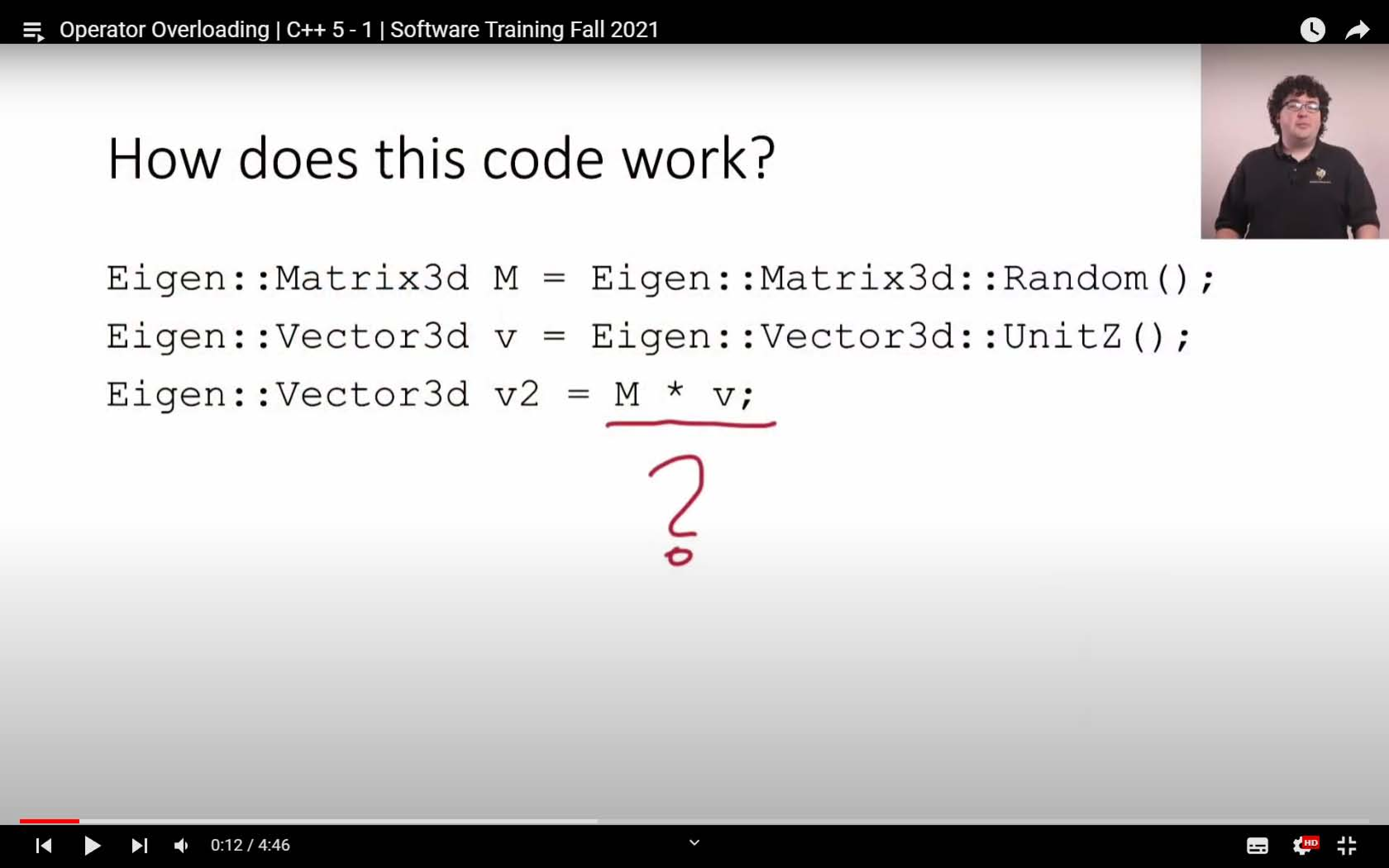


Operator Overloading | C++ 5 - 1 | Software Training Fall 2021

How does this code work?

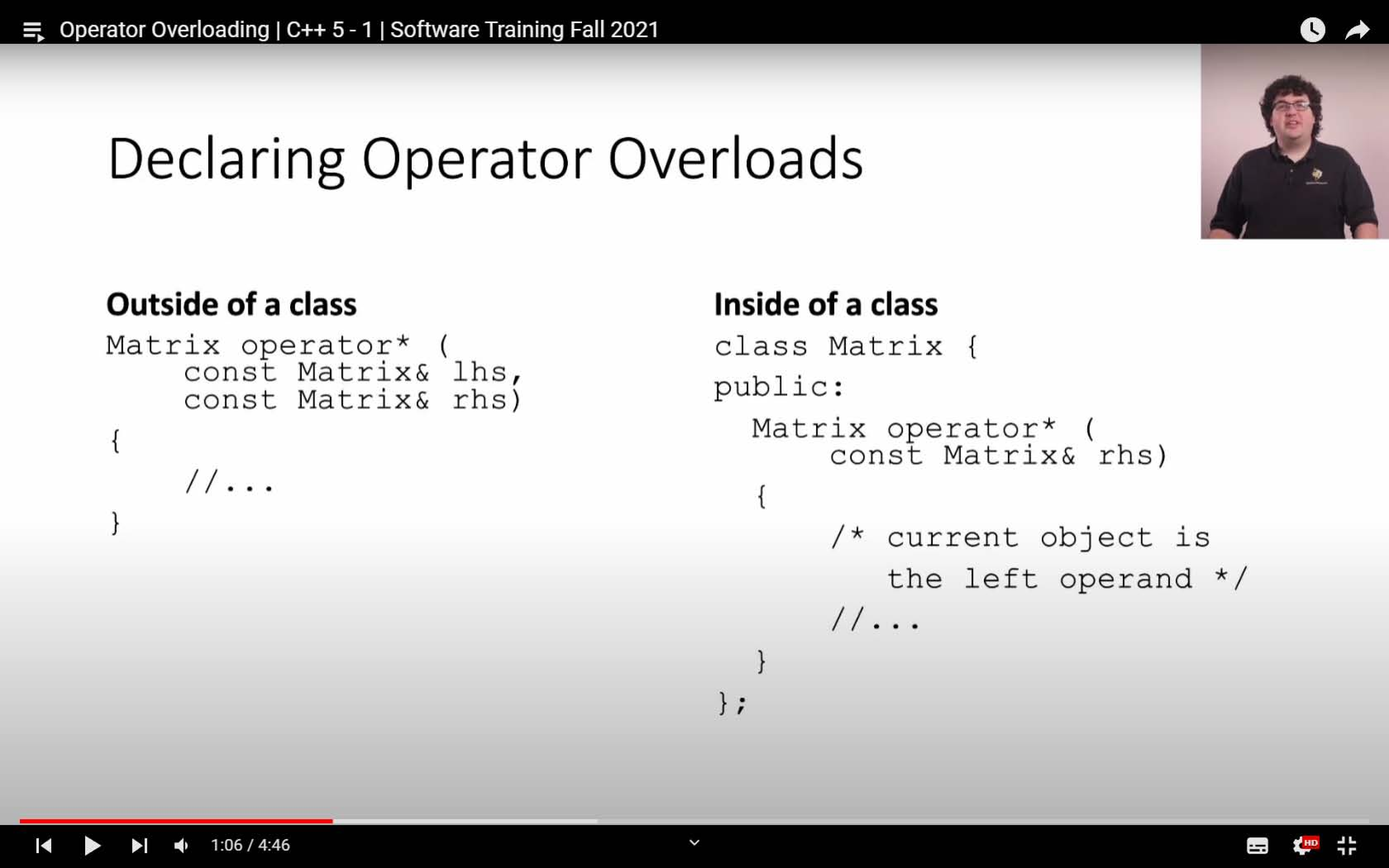
```
Eigen::Matrix3d M = Eigen::Matrix3d::Random();
Eigen::Vector3d v = Eigen::Vector3d::UnitZ();
Eigen::Vector3d v2 = M * v;
```

?

0:12 / 4:46

Operator Overloading

```
Matrix operator+ (const Matrix& lhs, const Matrix& rhs)
{
    //...
}
```



Operator Overloading | C++ 5 - 1 | Software Training Fall 2021

Declaring Operator Overloads

Outside of a class

```
Matrix operator* (
    const Matrix& lhs,
    const Matrix& rhs)
{
    //...
}
```

Inside of a class

```
class Matrix {
public:
  Matrix operator* (
      const Matrix& rhs)
  {
      /* current object is
         the left operand */
      //...
  }
};
```

1:06 / 4:46

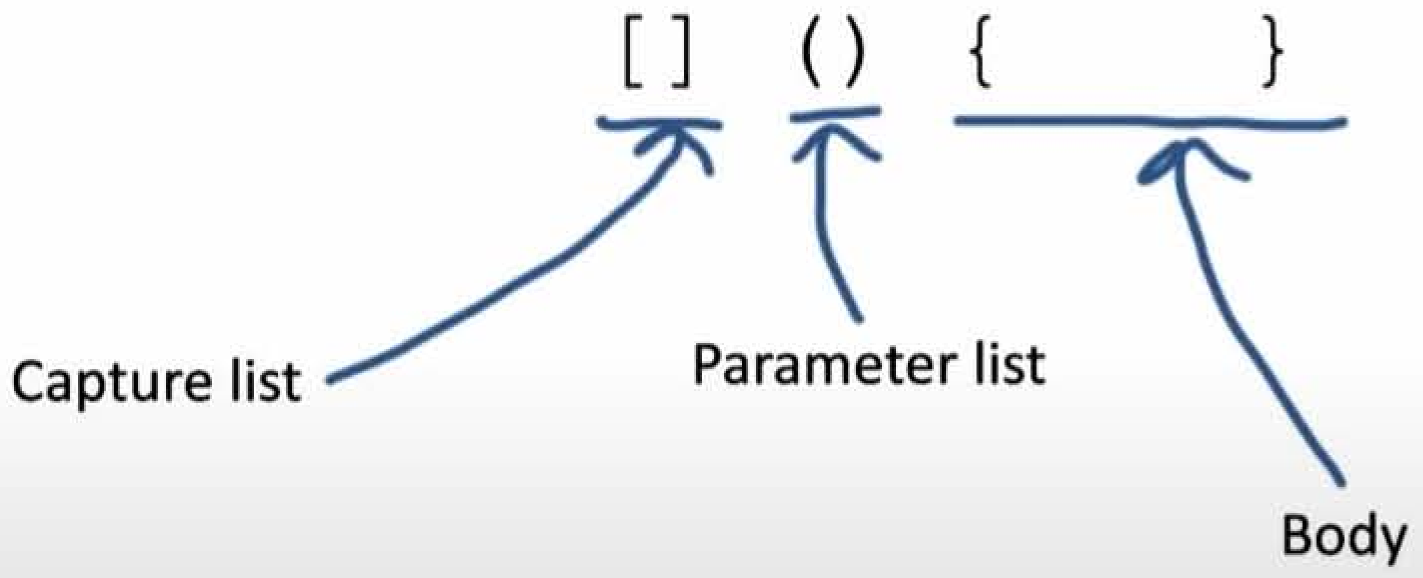
Callable Objects

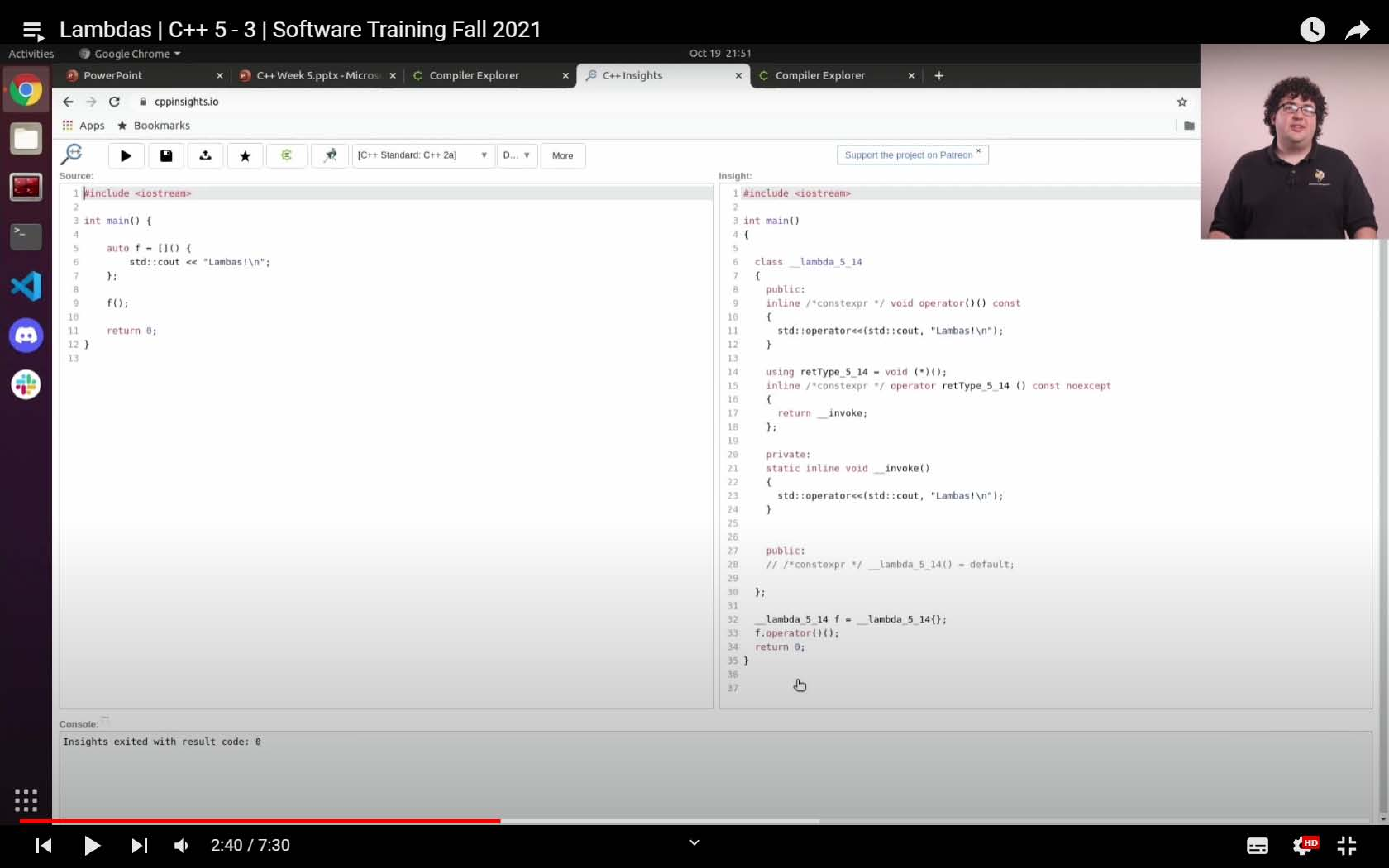
- Any object that overloads the "function call operator"

```
ReturnType operator() (const ParamType& p) {}
```

Lambdas

- Shorthand for creating an anonymous, callable object





Source:

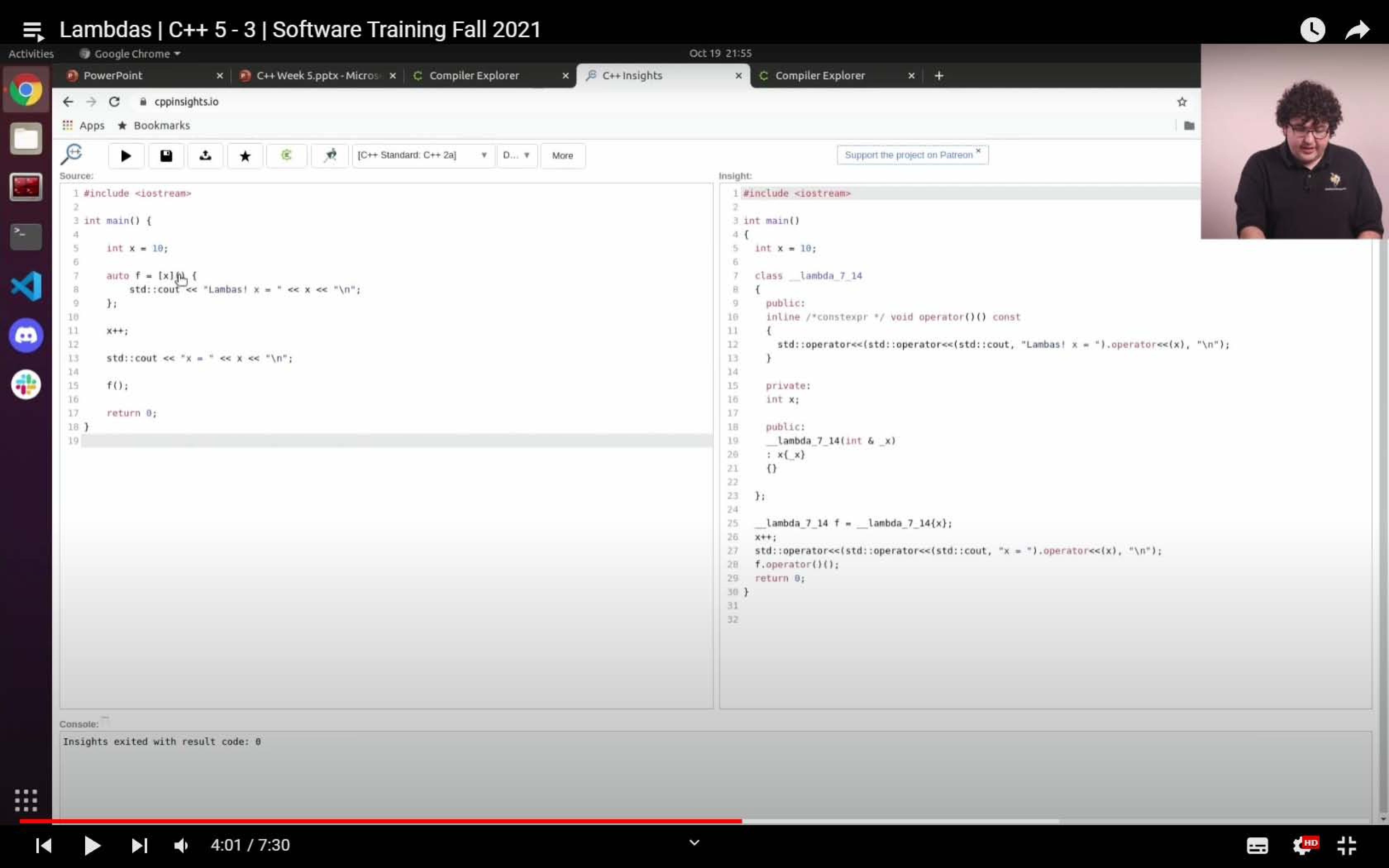
```cpp
#include <iostream>

int main() {

    auto f = []() {
        std::cout << "Lambas!\n";
    };

    f();

    return 0;
}
```

Insight:

```cpp
#include <iostream>

int main()
{

  class __lambda_5_14
  {
    public:
    inline /*constexpr */ void operator()() const
    {
      std::operator<<(std::cout, "Lambas!\n");
    }

    using retType_5_14 = void (*)();
    inline /*constexpr */ operator retType_5_14 () const noexcept
    {
      return __invoke;
    };

    private:
    static inline void __invoke()
    {
      std::operator<<(std::cout, "Lambas!\n");
    }


    public:
    // /*constexpr */ __lambda_5_14() = default;

  };

  __lambda_5_14 f = __lambda_5_14{};
  f.operator()();
  return 0;
}
```

Console:

```
Insights exited with result code: 0
```


Lambdas | C++ 5 - 3 | Software Training Fall 2021

Source:

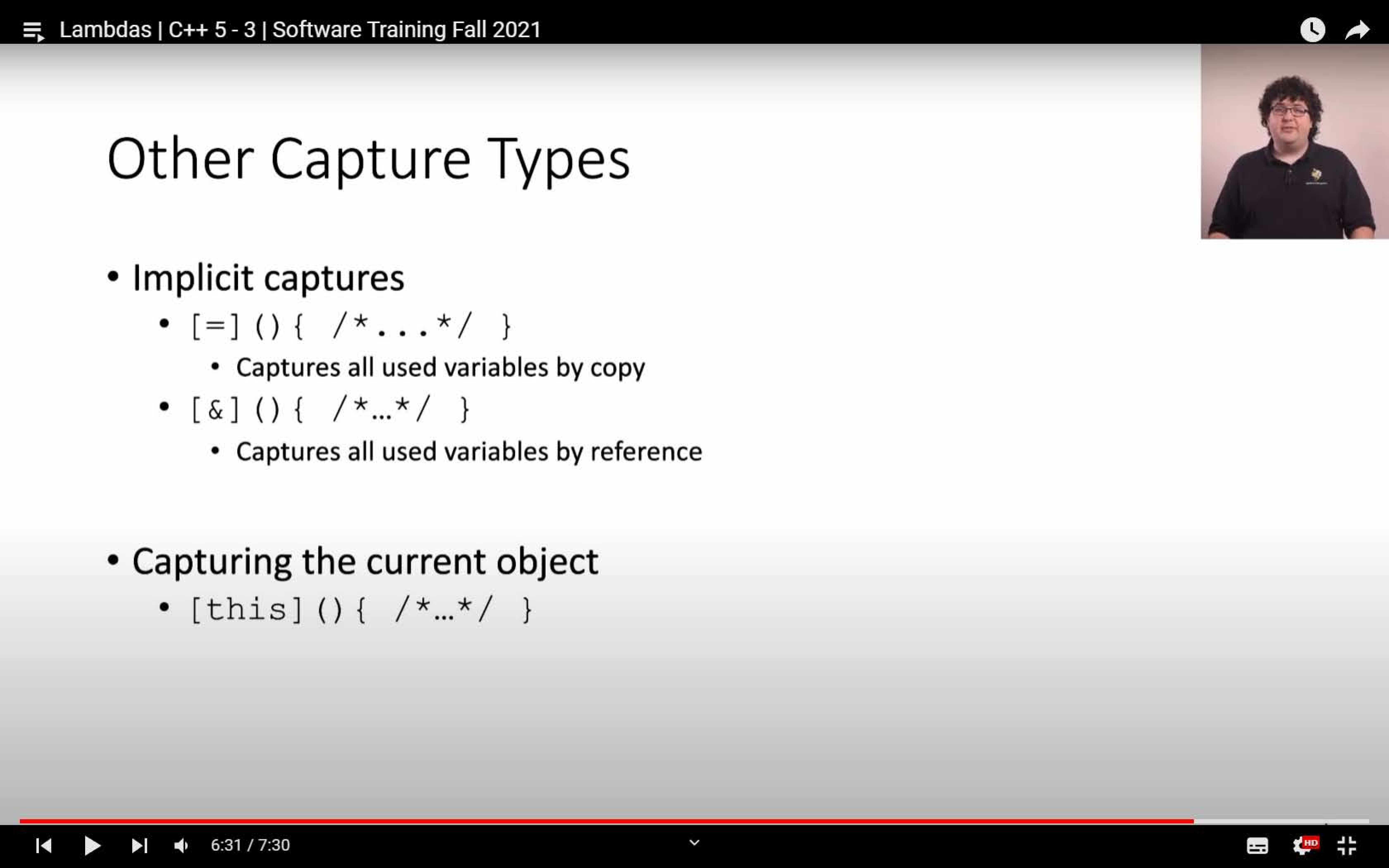
```cpp
#include <iostream>

int main() {

    int x = 10;

    auto f = [x]() {
        std::cout << "Lambas! x = " << x << "\n";
    };

    x++;

    std::cout << "x = " << x << "\n";

    f();

    return 0;
}
```

Insight:

```cpp
#include <iostream>

int main()
{
  int x = 10;

  class __lambda_7_14
  {
    public:
    inline /*constexpr */ void operator()() const
    {
      std::operator<<(std::operator<<(std::cout, "Lambas! x = ").operator<<(x), "\n");
    }

    private:
    int x;

    public:
    __lambda_7_14(int & _x)
    : x{_x}
    {}

  };

  __lambda_7_14 f = __lambda_7_14{x};
  x++;
  std::operator<<(std::operator<<(std::cout, "x = ").operator<<(x), "\n");
  f.operator()();
  return 0;
}
```

Console:

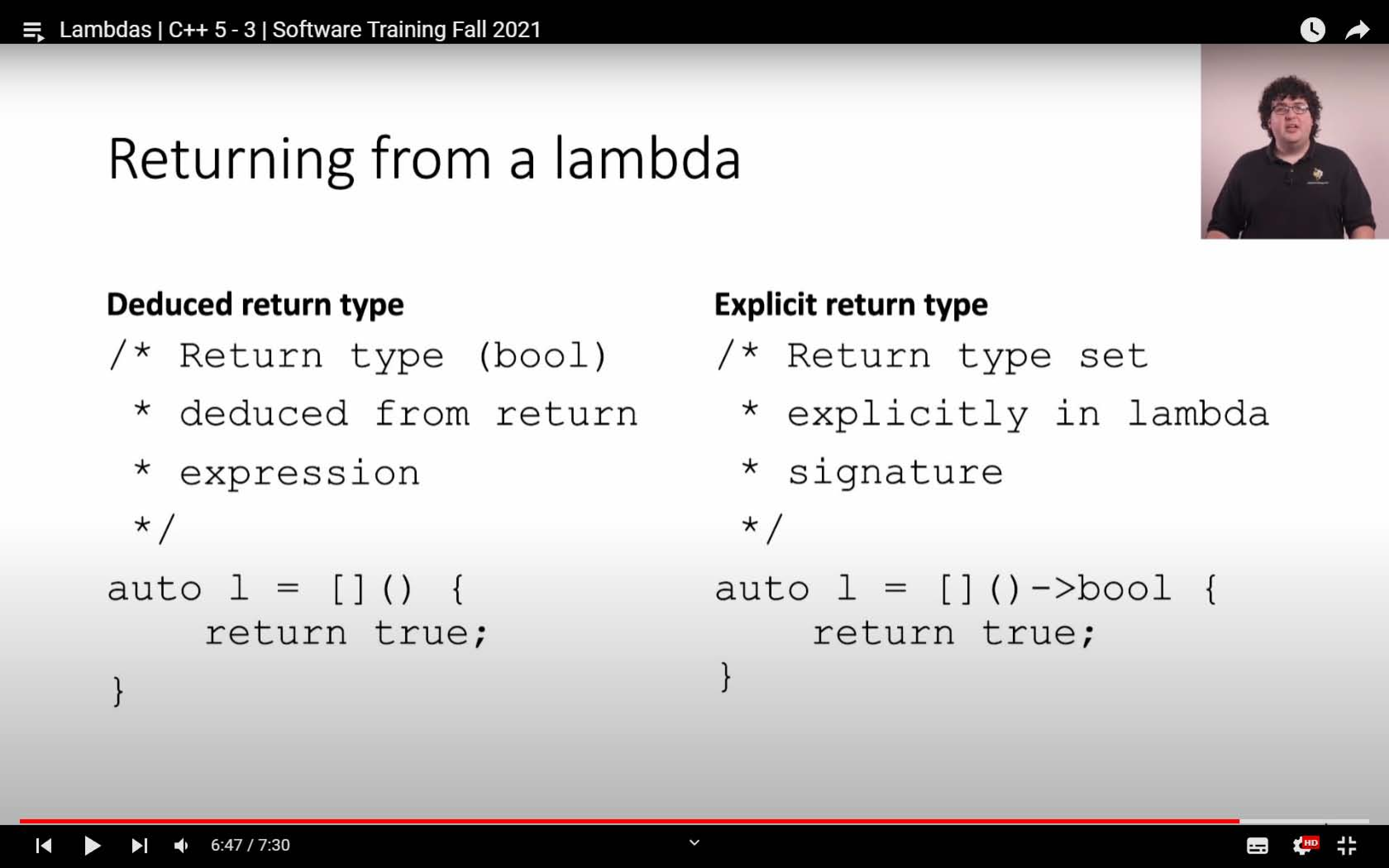
```
Insights exited with result code: 0
```



Other Capture Types

- Implicit captures
  - `[=](){ /*...*/ }`
    - Captures all used variables by copy
  - `[&](){ /*…*/ }`
    - Captures all used variables by reference

- Capturing the current object
  - `[this](){ /*…*/ }`



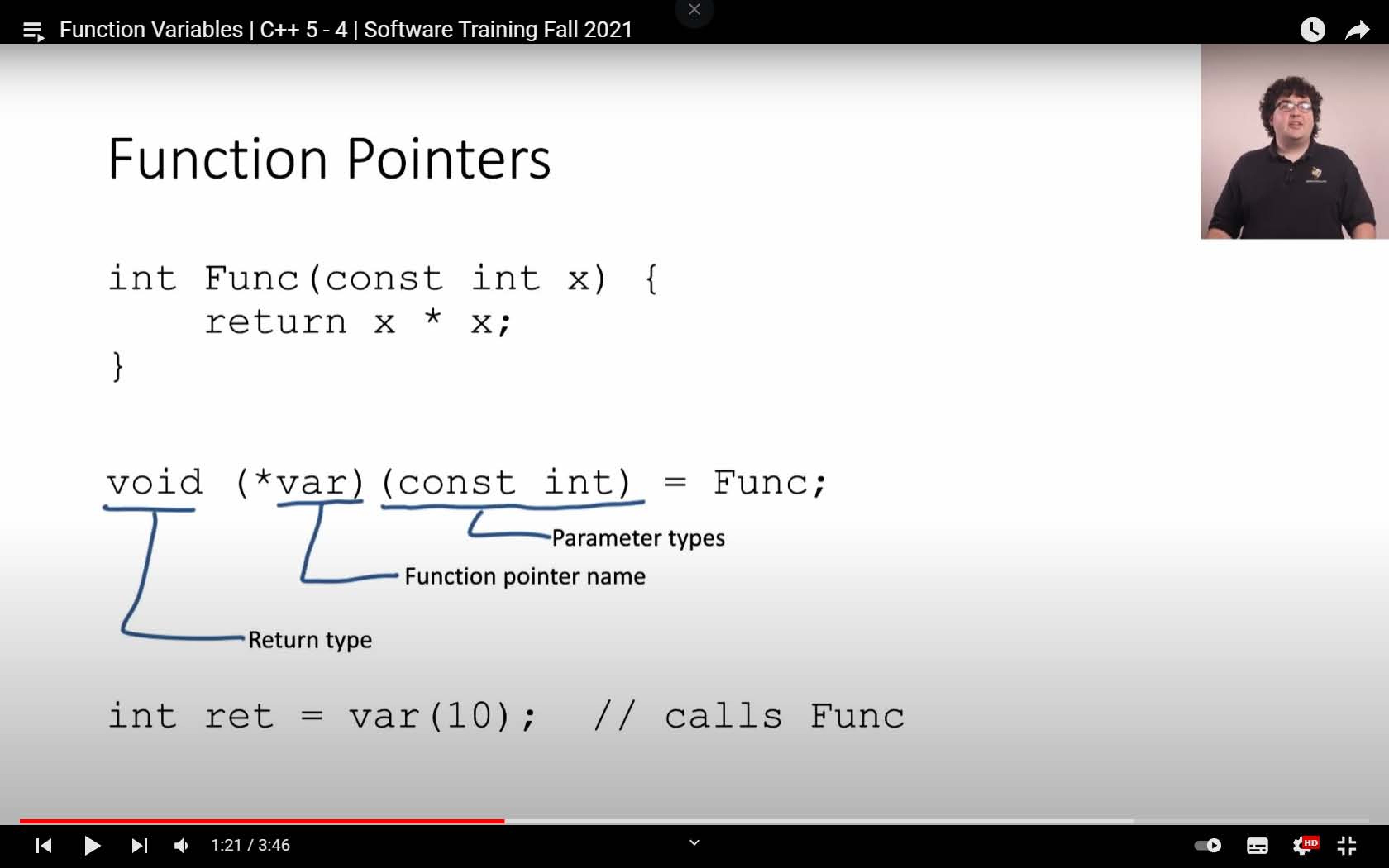
Returning from a lambda

Deduced return type

```
/* Return type (bool)
 * deduced from return
 * expression
 */
auto l = []() {
    return true;
}
```

Explicit return type

```
/* Return type set
 * explicitly in lambda
 * signature
 */
auto l = []()->bool {
    return true;
}
```


Function Variables | C++ 5 - 4 | Software Training Fall 2021
1:21 / 3:46

Function Pointers

```
int Func(const int x) {
    return x * x;
}


void (*var)(const int) = Func;
```

Parameter types
Function pointer name
Return type

```
int ret = var(10);  // calls Func
```

Function Variables | C++ 5 - 4 | Software Training Fall 2021

std::function

- Can hold a copy or reference to any callable thing
  - Functions
  - Lambdas (with / without captures)
  - Callable objects

```
std::function<ReturnType (ParamType p)> func_var;
```

2:14 / 3:46