Variables and Operators

Variables

- A name that can hold a value in our program
- Each variable holds a value of a specific type

Primitive Data Types

- Integral types
  - Whole numbers
- Floating point types
  - Fractional numbers
- Boolean values
  - True or false
- Characters
  - Letters / digits / symbols

Integral Type Sizes

- Keywords (see table)
- Fixed width types:
  - int8_t
  - int16_t
  - int32_t
  - int64_t

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
`short`	`short int`	at least **16**	**16**	**16**	**16**	**16**
`short int`						
`signed short`						
`signed short int`						
`unsigned short`	`unsigned short int`					
`unsigned short int`						
`int`	`int`	at least **16**	**16**	**32**	**32**	**32**
`signed`						
`signed int`						
`unsigned`	`unsigned int`					
`unsigned int`						
`long`	`long int`	at least **32**	**32**	**32**	**32**	**64**
`long int`						
`signed long`						
`signed long int`						
`unsigned long`	`unsigned long int`					
`unsigned long int`						
`long long`	`long long int` (C++11)	at least **64**	**64**	**64**	**64**	**64**
`long long int`						
`signed long long`						
`signed long long int`						
`unsigned long long`	`unsigned long long int` (C++11)					
`unsigned long long int`						

Signed vs Unsigned

- When using keywords, add the "unsigned" keyword
  - Ex: unsigned int
- When using fixed-width types, add "u" to the beginning
  - Ex: uint16_t

Floating Point Values

- float
  - 32 bits
- double
  - 64 bits
- long double
  - 128 bits

Character Types

- Used in older code to hold individual bytes (8 bits)
- char to hold ascii text
- wchar_t to hold multi-byte text encodings

Creating Variables

```
int count = 0;
```

type

name

initializer

Initialization

```
int a = 10;
int a(10);
int a{10};
```

Assignment Operator

```
int var1 = 10;  // initializes var1 to 10
var1 = 5;       // var1 now holds 5

int var2 = 20;  // initializes var2 to 20
var2 = var1;    // var2 now holds 5
```

Arithmetic Operators

```
a + b    // Addition
a - b    // Subtraction
a * b    // Multiplication
a / b    // Division
a % b    // Remainder
~a       // Bit-wise inversion
a & b    // Bit-wise AND
a | b    // Bit-wise OR
a ^ b    // Bit-wise XOR
a << b   // Shift a's bits left b positions
a >> b   // Shift a's bits right b positions
```

Assignment + Arithmetic Operators

```
a += b
a -= b
a *= b
a /= b
a %= b
a &= b
a |= b
a ^= b
a <<= b
a >>= b
```

```
int a = 2;
a *= 10; // a now holds 20
```

Increment / Decrement Operators

```
++a   // pre-increment
--a   // pre-decrement
a++   // post-increment
a--   // post-decrement

10++ -> 10
++10 -> 11
10-- -> 10
--10 ->  9
```

Using Multiple Operators

```
1 + 2 * 2   ->   5
```

```
(1 + 2) * 2   ->   6
```

Functions and Scope


Functions & Scopes | C++ 0 - 3 | Software Training Fall 2021
Functions
• A named block of code
• Defined inputs and outputs
• Useful for organizing and reusing code
RoboJackets

Functions in C++

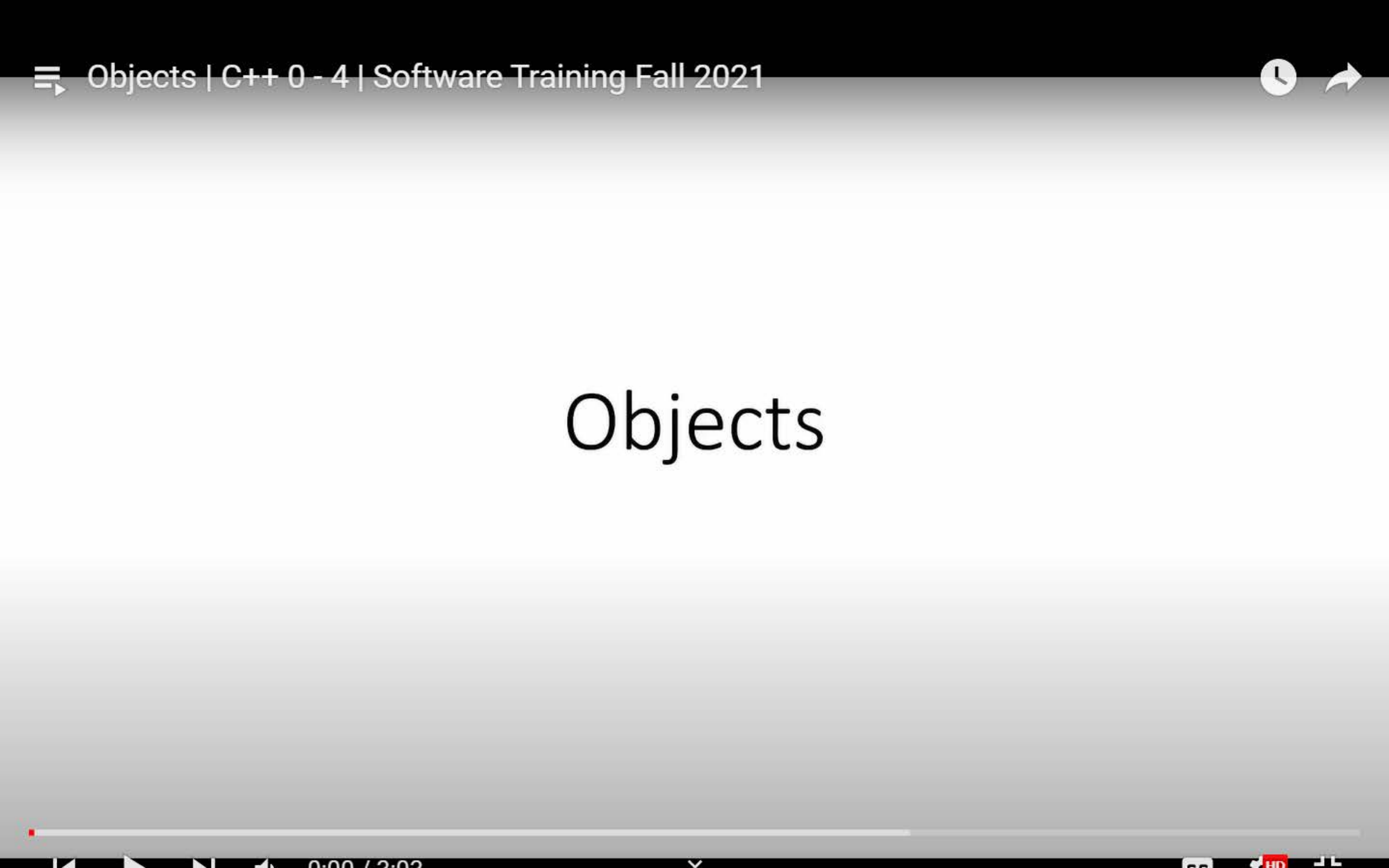
```
double GetAverage(double a, double b)
{
    return (a + b) / 2.0;
}
```

Returning Void

```
void ExampleFunction()
{
    // This function doesn't return anything
}
```

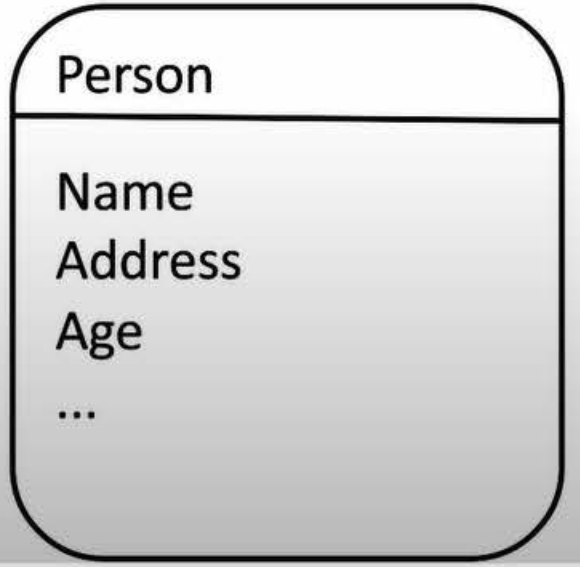
Scopes

```
void FuncA()
{
    int a = 10;
    // b doesn't exist in this scope
}

void FuncB()
{
    int b = 20;
    // a doesn't exist in this scope
}
```

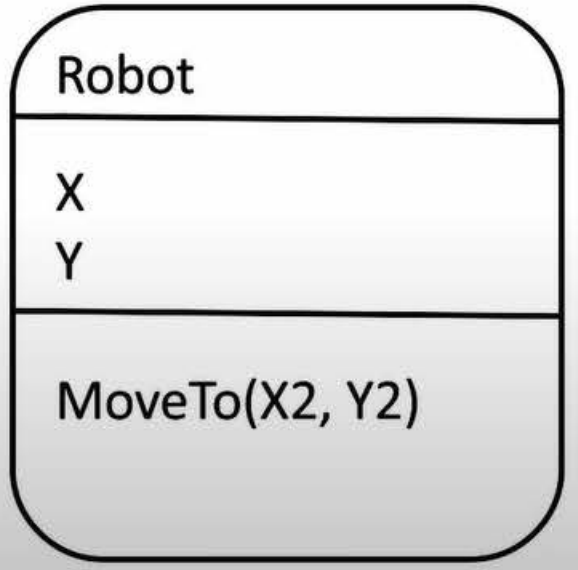

Objects | C++ 0 - 4 | Software Training Fall 2021
Objects

Objects

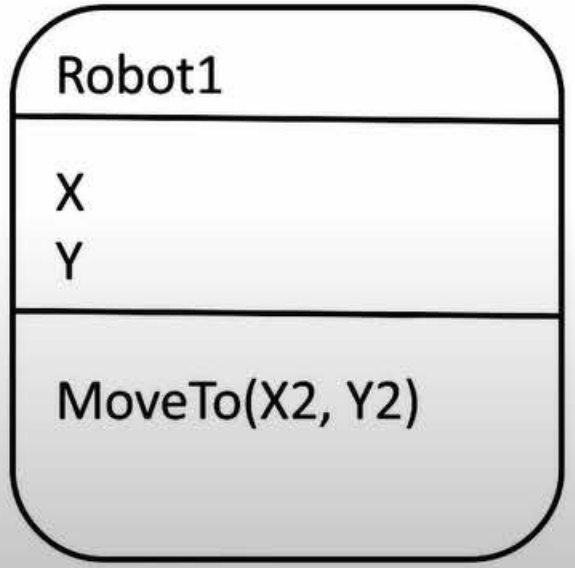
- Group data and behavior
- Objects contain member variables and member functions

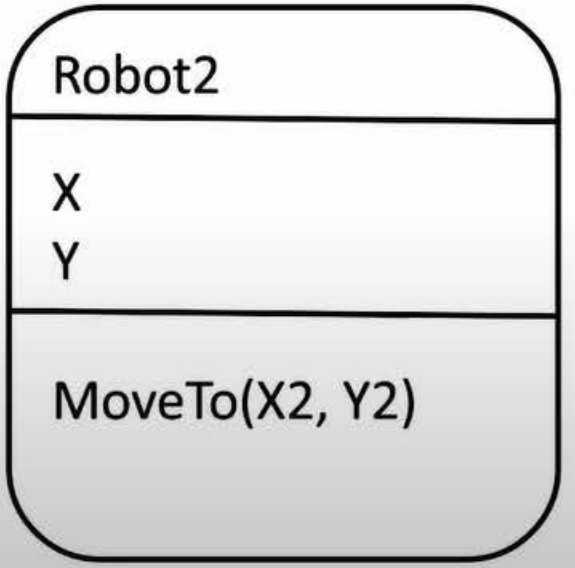


Objects



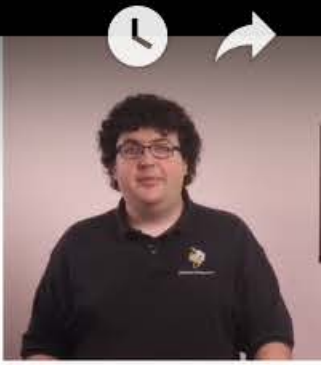
Objects





Classes

- Define the member variables and functions for a certain kind of object
- Are the types used with objects
- Each object is an instance of a class



Accessing Object Members

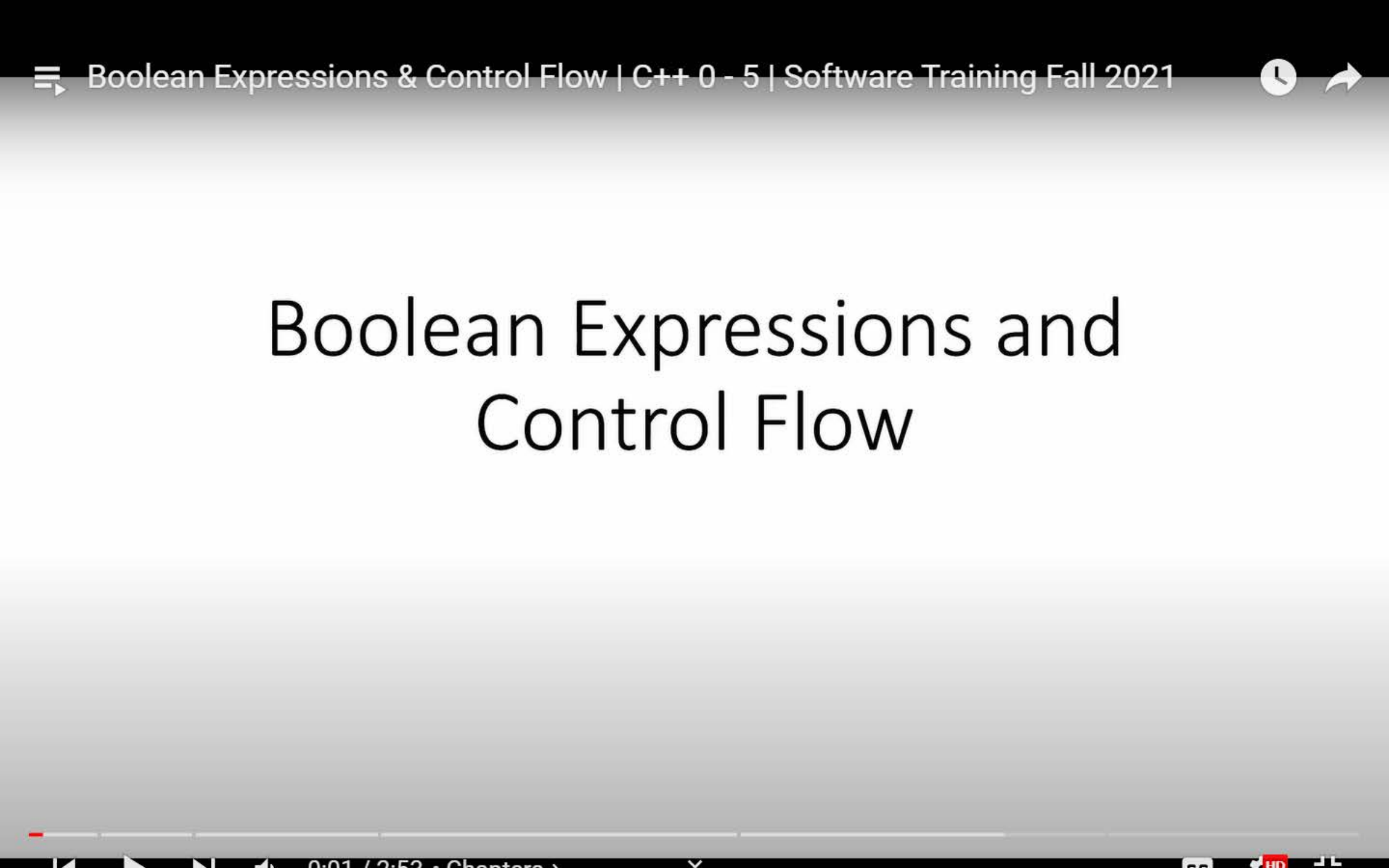
```
Person p1("Benjamin", 42);

p1.name  // gives us "Benjamin"
```

Constructors & Destructors

```
Person p1("Benjamin", 42);
```

- Constructors
  - Initialize objects
  - Called when you declare an object

- Destructors
  - Clean up objects
  - Called automatically when the variable's scope ends


Boolean Expressions & Control Flow | C++ 0 - 5 | Software Training Fall 2021
Boolean Expressions and
Control Flow

Boolean Expressions & Control Flow | C++ 0 - 5 | Software Training Fall 2021

Booleans

True or False

Comparison Operators

```
a == b    // equal to
a != b    // not equal to
a < b     // less than
a > b     // greater than
a <= b    // less than or equal to
a >= b    // greater than or equal to
a <=> b   // three-way comparison
```

Boolean Operators

```
!a       // negate value
a && b   // logical AND
a || b   // logical OR
```

If Statements

```
if ( (number % 2) == 0 ) {
  // this block executes when number is even
}
```

If Statements

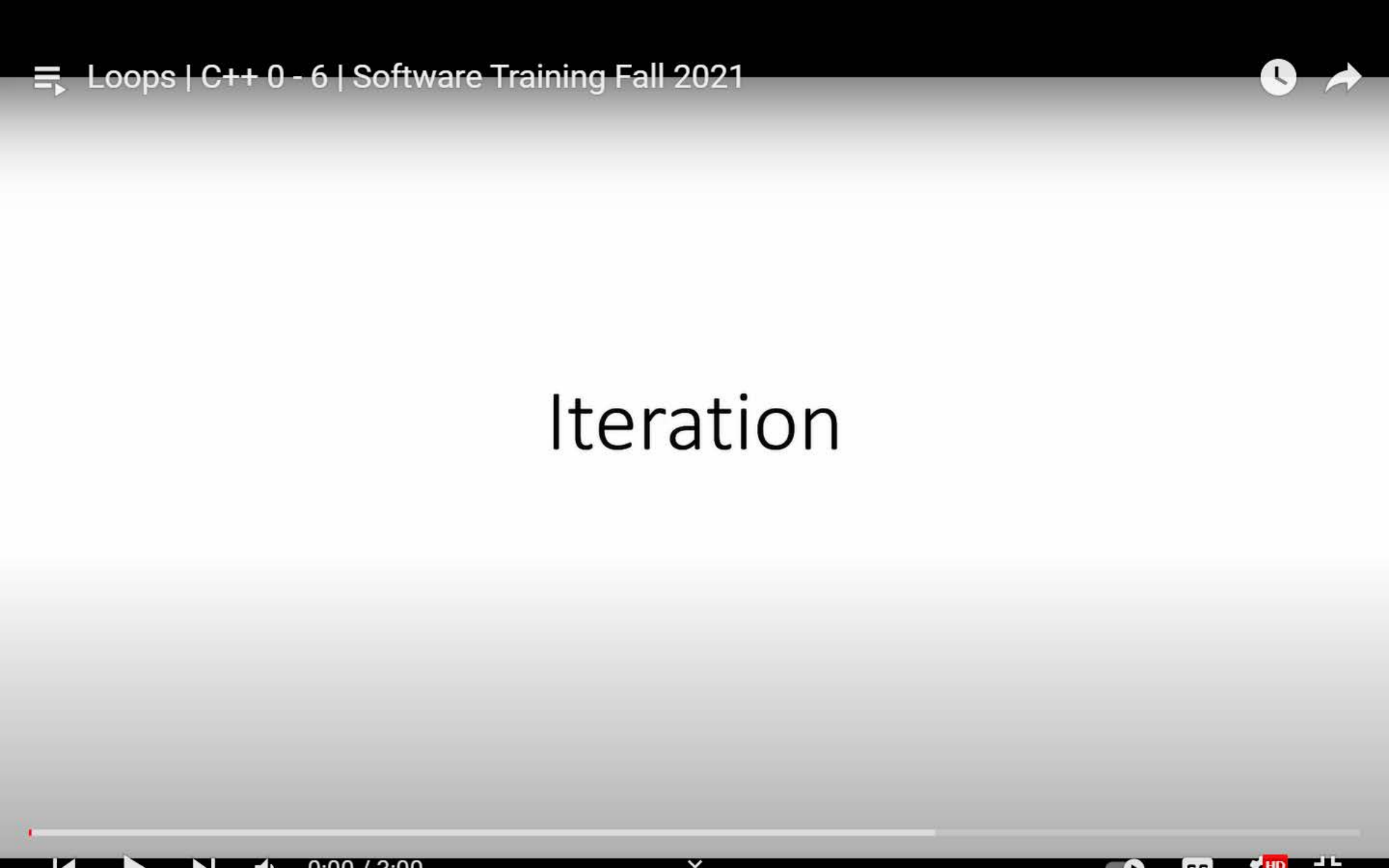
```
if ( (number % 2) == 1 ) {
   // this block executes when number is odd
}
else
{
   // this block executes when number is even
}
```

If Statements

```
if ( number < 0 ) {
   ...
}
else if ( number == 0 )
{
   ...
}
else
{
   ...
}
```

Switch Statements

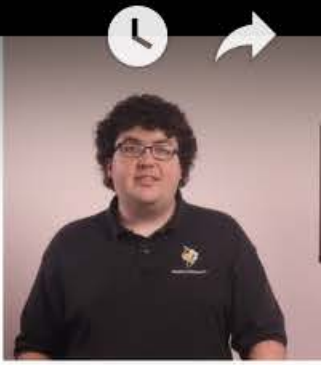
```
switch(count) {
  case 0:
    break;
  case 1:
    break;
  case 2:
    break;
  default:
    break;
}
```


Loops | C++ 0 - 6 | Software Training Fall 2021
Iteration

While Loop

```
int number = 100;
while ((number % 2) == 0)
{
    number /= 2;
}
```

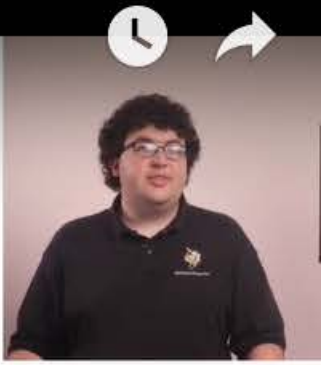
Loops | C++ 0 - 6 | Software Training Fall 2021



Counting With Loops

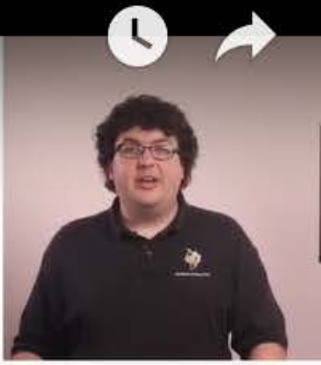
```
int i = 0;
while (i < 5)
{
    // This happens 5 times
    ++i;
}
```

0:59 / 3:09



For Loops

```
for(int i = 0; i < 5; ++i)
{
    // This happens 5 times
}
```



Control Flow and Scopes

- Bodies of control flow statements are scopes
- A new scope is created and destroyed for each iteration of a loop

```
for(int r = 0; r < table.rows; ++r)
{
  for(int c = 0; c < table.columns; ++c)
  {
    // Do something at position <r,c>
  }
}
```