Boolean evaluation and \( \texttt{if} \) statements

Making decisions in programs
By the end of this lesson you will be able to:

- Understand Boolean logic values
- Understand relational operators
- Understand `if` and `if/else` statements
- Understand the concept of blocks and scope
- Understand chaining and nesting of `if` and `if/else` statements
Why control?

- Most programs need to make a decision at some point
- Decisions in programs are similar to those we make every day
  - If it is sunny then I will go to the park, otherwise I will stay home
  - If I have money then I will go to the restaurant, otherwise I will cook at home
  - If my car has gas and it’s the weekend then I will go to on a road trip, but if my car has gas and it’s not the weekend then I will go to work, and last if my car does not have gas then I will go to the gas station
For making decisions we use **Boolean** logic in which expressions are evaluated as true or false

Logical expressions (e.g. if `age` is greater than or equal to 21, if `name` is equal to "Joe")

The basic data type, `bool`, can be used to store Boolean values but we typically do not need to declare variables of this type, instead opting to use the evaluation on the spot

- If you were printing out a user-input number multiplied by 2, you would not create a new variable for it, would you?
Relational operators

- These binary relational operators will help us compare two variables, values, or expressions to each other
  - `==` equal to (note this is very different from `=` which is assignment)
  - `!=` not equal to
  - `<` less than
  - `<=` less than or equal to
  - `>` greater than
  - `>=` greater than or equal to

- Expressions involving these operations evaluate to true or false
Understanding true and false

- Typically we use the decimal value of 1 to mean true and 0 to mean false.
- If you store true or false in a bool variable and output it using cout, it will display 1 or 0.
- In programming, we typically expand the definition to state that any non-zero value is true (e.g., 6 is true, 'C' is true, 0.0001 is true).
The `if` statement is used to execute a portion of code if an expression evaluates to true.

If the expression is not true then the code portion does not execute and the program continues.

On the next slide note:
- Parentheses around the expression
- Lack of a semicolon
- Indenting
if statement syntax

// single statement
if (expression)
    statement;

// multiple statements
if (expression)
{
    statement_1;
    statement_2;
    statement_3;
}

• If the expression evaluates to true then the statements will execute
Notes about braces

• if statements containing one statement within do not need braces, but it’s not syntactically wrong to have them:

```java
if (expression)
{
    statement;
}
```

• The portion of code denoted by braces is called a block
Blocks and scope

• Recall that the main body (main function) also uses braces – this is also a block

• Scope refers to the “visibility” of variables in code (i.e. where they can be used)

• You already know that variables can only be used after they are declared – we now add to that rule because of blocks
  – Variables that you declare **before** a block are accessible within the block
  – Variables that you declare **in** a block are only accessible within that block
  – You may reuse identifiers in blocks, but when accessed only the most **local** identifier is used
You have seen the most common style of indenting, called Allman style – I use it and the book uses it.

Allman style in a nutshell:

- Braces always go on a line by themselves (with few exceptions)
- Opening and closing braces always line up in the same column
- All code within braces is indented (tabbed over)

Indenting styles should be consistent and are used to assist in readability of programs.
A silly mistake

• For if statements containing multiple statements within, be sure to include braces or the result will be unexpected:

```java
if (expression)
    statement_1;
    statement_2;
    statement_3;
```

• `statement_2 and statement_3 will always execute!`
Evaluation of characters and numerical types

- Characters evaluate according to their ASCII values
- \( 4 \) is not equal to '4' (ASCII value 52)
- 'A' (65) is less than 'a' (97)
- All numerical types (floating point, integral, boolean, character) can be evaluated using relational operators to compare them to each other
Evaluation of strings

- Strings can only be evaluated with other strings (although you can evaluate an individual character like `name[0]` normally).
- A string is equal to another string when all the characters are identical.
- Strings are compared character-by-character, left-to-right according to the ASCII table.
- Shorter strings that are overlapped by a longer string are considered less than (e.g. "Joe" < "Joey" is true).
if/else statements

• If you want to decide between two portions of code to execute, you can use an if/else statement as such:

```java
if (expression)
    statement_1;
else
    statement_2;
```

• Either/both statements can be replaced by a block if multiple statements are needed
Chaining if/else

- You can chain as many `if/else` statements as you want:
  ```java
  if (expression_1)
    statement_1;
  else if (expression_2)
    statement_2;
  else
    statement_3;
  ```
- `statement_3` only executes if both `expression_1` and `expression_2` evaluate to false
Nesting if/else

- You can nest as many `if` and `if/else` statements as you want:

```java
if (expression_1)
{
    statement_1;
    if (expression_2)
        statement_2;
    statement_3;
}
```
• Write a program using `if/else` statements that has a menu and performs various activities (e.g. 1 for multiplication of two values, 2 for addition of two values, etc.)

• Use nesting with the above program to further prompt them to see if they want to multiply/add/etc. an additional value.