Lesson 3.09 — Relational Operators & if/else

Overview

Objectives — *Students will be able to...*

- **Evaluate** relational expressions
- **Predict and trace** the flow of an if statement.

Assessments — *Students will...*

- **Evaluate** relational expressions and **practice** correct if statement syntax during a game of grudgeball.

Homework — *Students will...*

- **Read** BJP 4.1 "Nested if/else" and "Object Equality"
- **Complete** self-check questions 7–9 and exercises 1 & 2

Materials & Prep

- **Projector and computer** (optional)
- **Whiteboard and markers**
- **Classroom copies** of Operator Precedence
- **Rules** for grudgeball (see website for details: http://toengagethemall.blogspot.com/2013/02/grudgeball-review-game-where-kids-attack.html)

Take the time to familiarize yourself with the rules of grudgeball, and test out your 2 and 3 point lines before class begins (you may need to readjust them). If you can get permission from your school to leave tape on the floor, it is helpful to have those lines down for the rest of the year. In future classes, if your students are having a hard time settling down during a review session, or can't stand a worksheet, you can always convert the worksheet or review session into a quick game of grudgeball.

Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction & note-taking	15min
Activity: Grudgeball	35min

Procedure

Rather than drill new rules with worksheets, the drilling/activity portion of the class will serve to tie the lesson together in the form of a class competition. If space and whiteboard setup allow, set up the grudgeball "court" and scoreboard before class begins so as to mystify the students.

Bell-work and Attendance [5 minutes]

Introduction & Note-Taking [15 minutes]

Before you begin lecture, announce to students that they should pay close attention, since the lecture content will be tested during the game.

- If you want to write code that executes some of the time, but not all the time, you can write an **if statement** (ask students what situations might use an if statement—if they are stuck, ask them to think about their last few programming projects!)

```
if (test) {          // Boolean Expression
    <statement>;     // Control Statement
```

```
    <statement>;    // Control Statement
}
```

- Have students come up to the front of the room to demonstrate the flow of control of the if control structure.
- Put 2 alternatives together by using an if/else statement:

```
if (test) {
    <statement>;
    <statement>;
} else {
    <statement>;
    <statement>;
}
```

- So what should students put in the "test" section to evaluate as true or false?

```
      3 * 3                ==            3 * 1 * 3
// <expression>  <relational operator>  <expression>
```

  Evaluate both expressions, then see if the relational operator holds true or not.

- Operators students need to know:
  - `==`: equal to
  - `!=`: not equal to
  - `<` : less than
  - `>` : greater than
  - `<=`: less than or equal to
  - `>=`: greater than or equal to |
- As far as precedence goes:
  - relational operators have a lower precedence than arithmetic operators
  - relational operators have higher than equality operators
  - inequality operators (`<`, `>`, `<=`, `>=`) have higher precedence than the equality operators (`==`, `!=`)

  `3 + 2 * 2 == 9` evaluates to: `false`, since `7` is not equal to `9`.

  Have students direct you how to solve this.

  If you'd like to refer to a visual aid for this segment of the introduction, poster 3.16.2 illustrates Java rules of precedence for all operators.
- You can only use a relational operator on primitive data types! (Ask students which types are included, which ones are excluded.)
- Review the mod % operator. Ask the students how they could use the mod operator to check for if a numbe is an odd or even number. Follow up with can a similiar algorithm be used to check if any number is evenly divisible by another.

Activity: Grudgeball [35 minutes] [Optional]

If you feel like your class has a grasp on the syntax and relational expressions, consider skipping this game and focusing on on-the-board examples (you can use the questions from Grudgeball below) or moving on to 3.10.

1. Divide students into their assigned teams.
2. Review the rules for grudgeball, and have the students repeat the rules back to you.
3. Using the problems listed below (and any you may add, depending on your class' needs), play grudgeball until a team wins, or until the class period ends.
    1. If a class gets the answer wrong, BRIEFLY pause the game to have students offer corrections before moving to the next team's question.

2. If correction seems to be dragging on, jump in and quickly re-teach using the incorrect answer as your example. It is important to keep the pace going to maintain student interest in the game!

4. Grudgeball problems & answers have been grouped assuming that you have 6 teams. If you have fewer teams, each "round" will be shifted accordingly, so you may have rounds where different teams are practicing different concepts. Judge each team's knowledge gaps, and adjust which questions you ask each group accordingly.

GRUDGEBALL PROBLEMS Translate these statements into logical tests that could be used in an if/else statement. Write the appropriate if statement with your logical test.

a) z is odd.
b) z is not greater than y's squre root.
c) y is positive.
d) Either x or y is even, and the other is odd.
e) y is a multiple of z.
f) y is a non-negative one digit number.

Given the variable declarations

```
int x =  4;
int y = -3;
int z =  4;
```

what are the results of the following relational expressions? (True or False?)

g) `x == 4`
h) `x == y`
i) `x == z`
j) `x + y > 0`
k) `y * y <= z`
l) `x * (y + 2) > y - (y + z) * 2`

Correct the following statement syntax errors:

m) `if x = 10 then {`
n) `if [x = 10] {`
o) `if (x equals 42){`
p) `if (x ==y){`
q) `If (x > 8){`
r) `IF(x <= 7){`

Identify and correct one of the (7) errors in the following code:

```
public class Oops4 {

    public static void main (String[] args) {
        int a = 7, b = 42;
        minimum(a,b);
        if {smaller = a} {
            System.out.println("a is the smallest!");
        }
    }

    // Returns the minimum of the parameters a and b.
    public static void minimum (int a, int b) {
        if (a < b) {
            int smaller = a;
        } else (a = b) {
            int smaller = b;
        }
```

```
        return int smaller;
    }
}
```

Accommodation and Differentiation

In ELL classrooms, you should read each question aloud in addition to showing it on the board or projector.

If this review session is too easy, give students time to start on the homework once you have finished Grudgeball.

Teacher Prior CS Knowledge

- The unary not operator `!` (commonly read as "bang") is commonly used in logical expressions to negate a boolean value. The use of not is unique among the logical operators as it takes only one operand. Not is analogous to the negative sign (`-`) where a minus before a numeric value or expression represents the negative of the value whereas a not (`!`) before a boolean value represents the opposite truth value: `true` becomes `false` and `false` becomes `true`.
- Programming languages have a default order of evaluating expressions commonly referred to as operator precedence. We usually teach students to be explicit in their logical expressions for clarity and to avoid mistakes. However, a knowledge of the languages precedence rules is valuable when both reading and debugging code. See the following for Java's precedence rules: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html. In order to grade correctly, it is essential to know Java's order of operations.

Common Mistakes

Conditionals common mistakes: http://interactivepython.org/runestone/static/JavaReview/Conditionals/cMistakes.html

Misconceptions

- Confusing the assignment operator (`=`) with the comparison operator (`==`) is one of the most common mistakes done by beginning programmers. Unfortunately, semantically a single equal sign is used to denote equality in mathematics and for most students they have been doing math much longer than computer science.

  It is important for the teacher when reading code aloud to differentiate single equals as assignment and double equals as comparison. The following would be read as:

  ```
  x = 5;      // "x is assigned 5"

  if (y == 4) // "if y is equal to 4"
  ```

- Many students think all statements in Java end in a semi-colon (`;`). However, this is not always the case as the if-block ends in curly braces when used with a block of code. Students in the habit of adding semi-colons to the end of each statement inevitably add semi-colons at the end of the conditional, like so:

  ```
  if (a > 1); // all statements end in semi-colon misconception
  ```

  When teaching students about Java statements, it is important to distinguish between statements that end in a semi-colon and those that signify scope with the curly-braces. Single Java statements such as declaring variables, assignment, etc. end in semi-colons while Java statements that denote scope like class definitions and methods have curly braces. If-statements can do both.

- Logical operators AND and OR are covered in a future lesson. However, students will combine logical operators similar to math syntax:

  ```
  if (9 <= grade <= 12) // relational operator misconception
  ```

  For students that ask about compound conditionals, you will need to defer and state they will be covered shortly in a future lesson. For students with prior programming experience inevitably searching for the correct syntax since the English words "and" and "or" do not work in Java.

- One common error is mismatched the parentheses:

  ```
  if (a == 10 // mismatched parentheses
  ```

  or mismatched curly braces:

  ```
  if (b == 12)
      dozen = true;
  }
  ```

  Even though many IDEs will help students by providing matching parentheses, a good habit for students to learn when writing code by hand is to add the closing parentheses right after they write the open parentheses. The same is true for open and close curly braces. Students will need to learn to write code by hand for the AP exam, so practice hand written Java is important to students' success.

  In addition, good coding style with proper indenting helps with student recognition of mismatched closing parentheses and curly braces because of the visual pattern. Good programming style makes it easier to identify errors in the pattern. This can be used during lab for students with mismatched parentheses and/or curly braces. Instead of find the error, asking the student to clean up their indenting will help them find errors on their own.

- When constructing if-statements, students insert the condition of the if-statement within the brackets instead of the parentheses:

  ```
  if {a > 1} // conditional vs block misconception
  ```

  See above "mismatched the parentheses" for good programming practice.

Video

- BJP 4–2, *Nested if/else Statements* (note: title of video is mislabeled, should be "if/else Statements") http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoPlayer.php?id=c4-2
- CSE 142, *if/else Statements* (38:41-50:15) https://www.youtube.com/watch?v=fo9_kOSs1Y8&start=2321
- CSE 142, *Methods with Conditional Execution* (4:08-18:39) https://www.youtube.com/watch?v=0mqNzejWfSY&start=248
- CS Homework Bytes, *Relational and Conditional Operators, with Elizabeth* https://www.youtube.com/watch?v=M-mYpnPygY0
- CS Homework Bytes, *If-Statements, with Jim* https://www.youtube.com/watch?v=SxWFYfA4i0M

Forum discussion

Lesson 3.09 Relational Operators & if/else (TEALS Discourse account required)