Lab 04: Booleans, predicates, conditionals

Create a separate file for each question. Keep them in your "Labs" folder, with the name `liiqj` for Lab ii, Question j. Download the headers for each function from the file `labinterface04.rkt` linked off the "Labs" page on the course Web site.

After you have completed a question (except class exercises), including creating tests for it, you can obtain feedback by submitting it and requesting a public test. Follow the instructions given in the Style Guide.

Language level: Beginning Student.

1. *[Class exercise with lab instructor assistance]* Create a function *two-multiples* that consumes three numbers, *target*, *candidate1*, and *candidate2*, and determines whether *target* is a multiple of both candidates. Your function should produce "both" if it is multiples of both, "neither" if it is a multiple of neither, and the value of the candidate if it is the multiple of one of the two. If any of the three numbers is a non-integer, your function should produce *false*.

2. [Modified from HtDP exercise 4.2.1] Translate each of the following subsets of the real line into Scheme functions that consume a number and produce *true* if the number is in the subset and *false* if it is outside the subset: (3,7]; the union of (1,3) and (9,11); the range of numbers outside of [1,3] (use the names *in-subset-1?, in-subset-2?*, and *in-subset-3?*). For example (in-subset-1? 2) produces false since $2 \leq 3$.

3. Consider an auction where the rules are such that each new bid must be at least 5% higher than the current high bid. For example, if the current high bid is $100, then the next bid must be at least $105. Create a function acceptable-bid? That consumes two positive numbers (*current-high* and *next-bid*) and produces true if next-bid includes an increase of at least 5% when compared to *current-high*, and false otherwise. Try writing the body of the function without using a cond expression.

4. Create a function new-string that consumes two strings (called *original* and *add-on*) and a symbol (called *position*), and produces the string *original* followed by *add-on* if *position* is *'after*, and the string *add-on* followed by *original* if *position* is *'before*, and produces *original* for any other value of *position*. For example, (new-string "abc" "123" 'before) produces "123abc" and (new-string "abc" "123" 'after) produces "abc123".

5. Create a function *switch-case-char* that consumes a character and produces the same character but with switched case. That is, the character should be produced in upper-case if it is a lower-case letter, and in lower-case if it is an upper-case letter. If the consumed character is not a letter, reproduce the same character. You may wish to consult the supplementary information on strings, linked off the Resources page on the course Web site.

6. Create the function *connect?* that consumes two strings and determines if the last letter in the first string is the same as the first letter in the second string.

7. *Optional open-ended questions* You can now refine your Pig Latin, comparative, and superlative functions by handling cases differently depending on the starting letter or letters (for Pig Latin) and the ending letter or letters (for comparatives and superlatives).

Helpful tips

Highlighted unused code After you have run your program, any unused part of the code will be highlighted. This either means that you have parts of the code that are not needed (and should be removed) or that you need to add more tests.