Lab 2

This week's lab will help you to practice:

- conditionals
- function composition: writing more complex functions by combining simpler functions
- using the design recipe

Part 1

In this part of the lab, you will practice writing functions that involve conditional expressions.

Problem 1

Design a function to compute monthly updates for bank accounts. For this problem, a bank account is simply a number representing the balance. The monthly update is not the same for every balance. For accounts with a negative balance the bank charges a $20 overdraft fee. For accounts with a balance of less than $20 the bank charges a low balance fee of $1. Accounts with a balance of $100,000 or more earn a 1% interest credit. All other accounts earn a 0.1% interest credit. Write the function `monthly-update` which consumes a balance and produces the new balance after the fee or credit is applied. Use constants where it is logical to do so.

Problem 2

Write the function whats-it-like-out which consumes a number for temperature, and produces a string. This function gives an evaluation of the current outdoor temperature.

For temperatures [0-40) degrees it should produce "cold"

For temperatures [40-60) degrees it should produce "chilly"

For temperatures [60-70) degrees it should produce "cool"

For temperatures [70-80) degrees it should produce "nice"

For temperatures [80-90) degrees it should produce "warm"

For temperatures [90-100] degrees it should produce "hot"

For temperatures above 100 or below 0 degrees it should produce "stay inside"

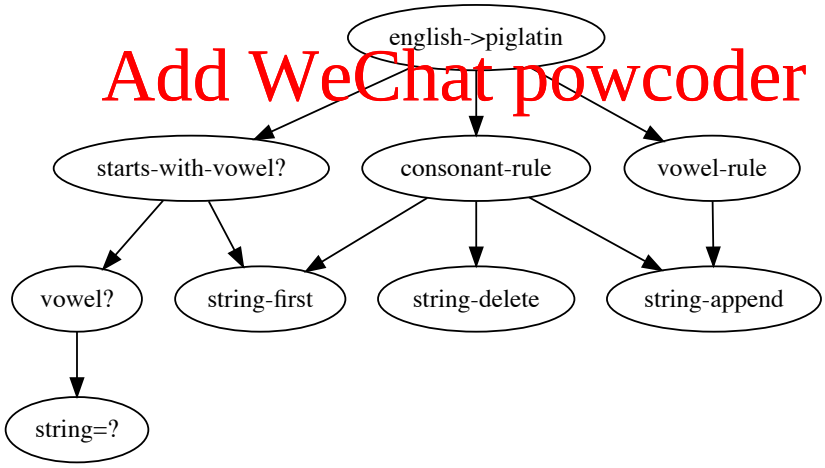
*note closed braces "[" denotes inclusive and open braces "(" denotes exclusive.

Part 2

In this part of the lab, you will create a small program for translating English words into a simplified version of Pig Latin. Because this problem is fairly complex, you will need to follow one of the most important aspect of program design: one task, one function, one purpose. That is divide one large problems into several smaller problems then solve the smaller problems (subdividing again, if necessary) and compose the solutions together to generate the overall results. In most cases, you'll need to tackle the dividing yourself, but in this case, to help you practice, we'll provide some strong hints.

At a high level, the translation process works as follows: if the given word starts with a vowel (a, e, i, o, or u), the Pig Latin version is the given word with the string "ay" added to the end (e.g., "omelet" --> "omeletay"). Otherwise, if the given word starts with a consonant, the Pig Latin version is the given word where the initial consonant is moved to the end of the word and then "ay" is added (e.g., "banana" --> "ananabay").

Notice that this large problem can be divided into (at least) three smaller problems: 1) determining whether the given word starts with a vowel or consonant; 2) applying the vowel rule; and 3) applying the consonant rule. These resulting problems can be further subdivided into even smaller problems. For example, determining whether a word starts with a vowel requires 1) a function to extract the first character of the word and 2) a function to check whether a given character is a vowel or not. In addition, the consonant and vowel rules also require a function for getting the first character of a string as well as functions for adding and deleting characters from a given string. The full decomposition is shown in the figure below.



In the graph, each node (circle) is a function and edges indicate the calling relationship (e.g., `english->piglatin` calls `starts-with-vowel?`, `consonant-rule` and `vowel-rule`). Note that the leaf nodes are either functions that DrRacket provides as part of the standard library (i.e., `string-append`, `string=?`) or functions that you have already written (i.e., `string-first`, `string-delete`). This is exactly what you want to have, the problem has been fully decomposed when the smallest problems can be trivially solved by calling one function.

After deciding what functions are necessary, the next step is to design them following the Design Recipe. Bottom-up is a good approach for this. Start with the leaves (simplest functions) because you already have everything you need to write them. In this case start with, `vowel?`, `consonant-rule`, and `vowel-rule`. Then write `starts-with-vowel?`. Finally, design `english->piglatin`. Note that you may copy your implementations of `string-first` and `string-delete` from Lab 1.