Assignment: 3

Due: Thursday, June 4 at 9:00 am
Language level: Beginning Student
Coverage: Modules 1–3

For this and all subsequent assignments, to receive full marks you are required to use the design recipe for every function you write. You may include the examples given in the assignment in your submissions, but they will be ignored by the markers—you must develop a complete suite of examples and tests on your own. For your convenience, an interface file which contains the headers of the required functions is available on the course webpage.

Do not send any code files to course staff; they will not be accepted. Submissions must be made via MarkUs as described on the course webpage. After submission, check your basic test results to ensure your files were properly submitted. Solutions that do not pass the basic tests are unlikely to receive any correctness marks.

Remember, the solutions you submit must be **entirely your own work**.

1. Recall the *pig-latin* function written in assignment 2. It followed a simple conversion scheme which didn't work well for many words. For example, `"argument"` would translate to `"rgumentaay"`, which is hard to pronounce.

   For this question, write an improved *pig-latin* translation function which consumes a string *word* and produces the translation of *word* into Pig Latin, using the following rules:

   - If the first letter of the word begins with a vowel (a, e, i, o, u) then add "way" to the end of the word. For example, (*pig-latin* `"argument"`) should produce `"argumentway"`.
   - Otherwise, remove all the letters until the first vowel, add them to the end of the word, followed by "ay". For example, (*pig-latin* `"scheme"`) should produce `"emeschay"`.
   - The letter y should be treated as a vowel in the second case (when it appears after the first character) but not in the first case. For example, (*pig-latin* `"syntax"`) should produce `"yntaxsay"` and (*pig-latin* `"you"`) should produce `"ouyay"`.

   You can assume that all characters of *word* are lower-case characters of the alphabet and that at least one vowel appears in the first four characters of *word*.

2. Your WatIAM userid is typically a combination of initials from your given names, possibly followed by some digits, followed by your surname (possibly truncated). For example, Jonathan William Smith could have a userid of `jw25smit`.

   Write a Racket function *create-userid* which consumes three strings representing a student's name (*first-name*, *middle-name*, and *last-name*, in that order) and a positive natural number *n* representing the digits which appear in the student's userid. The function should produce the userid for the student whose information was consumed by the function, taking into consideration the following caveats:

   - The student may not have a middle name. In this case, *middle-name* will be the empty string and the initials portion of the userid will only consist of the first letter of the student's first name. Otherwise, the initials portion of the userid will consist of the first letter of *first-name* followed by the first letter of *middle-name*.
   - If *n* is 1 then no digits should appear in the userid.
   - The student's userid must have at most 8 characters; if the string containing the student's initials, userid digits, and last name is longer than 8 characters then the last name is truncated so that the userid has exactly 8 characters.

   You can assume that the given strings only consist of lowercase alphabetic characters, that *first-name* and *last-name* are nonempty, and that *n* is between 1 and 999.

   For example, (*create-userid* `"jonathan" "william" "smith" 25`) should produce `"jw25smit"`.

   Hint: The function *number→string* will be helpful (see the Racket documentation).

3. When represented on a seven-segment display, some digits still look like digits when turned upside-down. In particular, when turned upside-down the digits 0, 1, 2, 5, and 8 look like the same digit, while 6 looks like 9 and vice-versa.

   (a) Write a Racket function *upside-down-number?* which consumes a natural number *n* with exactly 4 digits (i.e., between 1000 and 9999) and produces *true* if it still looks like some number when turned upside-down and *false* if it contains a digit which doesn't look like a number when turned upside-down (i.e., *n* contains at least one of the digits 3, 4, or 7). For example, (*upside-down-number?* 5678) should produce *false*.

   (b) Furthermore, the remaining digits still look like certain characters when turned upside-down; namely, 3 looks like the letter E, 4 looks like the letter h, and 7 looks like the letter L. Write a Racket function *turn-upside-down* which consumes a natural number *n* with exactly 4 digits (i.e., between 1000 and 9999) and produces a string containing a representation of what *n* looks like upside-down, using the translations given above.

   Note that the order of the digits in *n* become reversed when turned upside-down. For example, (*turn-upside-down* 1234) should produce `"hE21"`.

   Hint: The function *number→string* may be helpful. You could also consider using an *extract-digit* helper function, reusing your code from assignment 2 question 3 (b).