HW_04-Racket_lists

Actividad # 4

Lenguajes de programación

A01703613 - Daniel Cu Sánchez

Instructions

1. Write a Racket script that contains the following functions:
    1. **hailstone**: The function takes an initial integer as argument, and will generate a list of numbers, starting at the number provided and finishing at 1. The numbers in between are obtained and added to the list as follows:
        1. If the number is even, divide it by 2
        2. If the number is odd, multiply it by 3 and add 1
    2. **hailstone-list**: The function takes two integers, one lower limit, and an upper limit. It will generate a new list with the results of the hailstone function from the lower to the upper limits.
    3. **shift-char**: Takes a character and an integer. If the character is a letter, either lowercase or uppercase, it will change the character for the one at the distance indicated by the integer. Other characters different from letters will be left as they are. The integer can be positive, negative or zero. It should wrap around the letters in the English alphabet. The existing functions *char->integer* and *integer->char* can help here. This function does not need to use lists, but will be necessary for the next function.
    4. **caesar-encode**: Takes a string, an integer and a boolean. If the boolean value is **false**, the function will encode the string by shifting each of its letter characters the distance indicated by the integer. If the boolean is **true**, the function will decode the string, by using the negative of the integer. Characters that are not letters should not be changed. The functions *string->list* and *list->string* can help.

```
#|
    hailstone: The function takes an initial integer as argument,
    and will generate a list of numbers, starting at the number provided and finishing at 1.
    The numbers in between are obtained and added to the list as follows:
    If the number is even, divide it by 2
    If the number is odd, multiply it by 3 and add 1

|#

;Receiving an initial int number
(define (hailstone number)
    (let loop
        ;Declaring initial values
        ([value number] [result empty])
        ;Checking if the initial number is zero
        (if (= value 0) result
            ;Checking if a list is not created
            (if (empty? result)
                ;Creating a list with the number as a initial value
                (loop value (list (append result value)))
                ;Verifying the end loop
                (if (= value 1) result
                    ;Checking if the next value on the list is odd or even
                    (if (even? value)
                        ;Even
                        (loop (/ value 2) (append result (list (/ value 2))))
                        ;Odd
                        (loop (+ (* value 3) 1) (append result (list (+ (* value 3) 1))))
                    )
                )
            )
        )
```

```
#|
    hailstone-list: The function takes two integers, one lower limit, and an upper limit.
    It will generate a new list with the results of the hailstone function from the lower to the upper limits.
|#


;Receiving lower and upper limits
(define (hailstone-list low-limit up-limit)
    (let loop
        ;Start from lower limit
        ([a low-limit] [result-value empty])
        ;In case has not reached the upper
        (if (≤ a up-limit)
            ;Creating lists by using hailstone function from exercise one
            (loop (+ a 1) (append result-value (list (hailstone a))))
            result-value
        )
    )
)
;This function return a list
```

```
(define (shift-char character-symbol quantity)
    ;Starting tail recursion
    (let loop
        ([ch-sym character-symbol] [num quantity])
        (cond
            ;Checking if character-symbol is between 64 and 91
            [(and (≥ (char→integer ch-sym) 65) (≤ (char→integer ch-sym) 90))
                (cond
                    ; Converting character-symbol if the sum is 90
                    [(> (+ num (char→integer ch-sym)) 90)
                        ;Calling recursion, character-symbol is converted to char
                        (loop (integer→char 65)
                            (- (+ num (char→integer ch-sym)) 91))]
                    ;Converting character-symbol to int if sum less than 65
                    [(< (+ num (char→integer ch-sym)) 65)
                        (loop (integer→char 65)
                            ;Converting character-symbol to int if sum num and character-symbol is 65 the substract
                            (- 26 (- 65 (+ num (char→integer ch-sym)))))]
                    [else (integer→char (+ num (char→integer ch-sym)))])]
            ;if character-symbol is between 96 and 123
            [(and (≥ (char→integer ch-sym) 97) (≤ (char→integer ch-sym) 122))
                (cond
                    ;ch-sym to int if sum greater than 122
                    [(> (+ num (char→integer ch-sym)) 122)
                        ;Recursision ch-sym to char
                        (loop (integer→char 97)
                            (- (+ num (char→integer ch-sym)) 123))]
                    ;ch-sym to int
                    [(< (+ num (char→integer ch-sym)) 97)
                        ;Recursive 97 to ch-sym
                        (loop (integer→char 97)
                            (- 26 (- 97 (+ num (char→integer ch-sym)))))]
                    [else (integer→char (+ num (char→integer ch-sym)))])]
            [else ch-sym])))
;This function return a char
```

```
#|
caesar-encode: Takes a string,  an integer and a boolean. If the boolean value is false, the function will encode the strin

Characters that are not letters should not be changed. The functions string→list and list→string can help.
|#

; Recieving a string,integer and a boolean.
(define (caesar-encode string number boolean)
        ;Helper function, this calls shift-car
        (define (helper-func string-help num-help m)
            (let loop ([s-list (string→list string-help)] [n num-help] [ans empty])
                (if (empty? s-list)
                    (list→string ans)
                    (loop (cdr s-list) n (append ans (list (shift-char (car s-list) (* n m)
                )
            )
        )
    ))))
    (if (equal? boolean #t)
        (helper-func string number -1); If boolean true, the function will decode the string by using the negative of the i
        (helper-func string number 1)
    )
); In the other hand the function will encode the string by using shifting.

;This function return a string
```

Run command

(load "hw-4.2-test.rkt")