```
#|
Fourth Racket Homework

Rodrigo Benavente García
22/09/21
|#

#lang racket

(define (hailstone num);First exercise, recieves an initial int
  (let loop ;Start an anonymous function
    ([a num] [result empty]);Declare initial values
    (if (= a 0) result;We check if the initial num isn't zero
        (if (empty? result);Checks if a list hasn't been created yet
            (loop a (list (append result a)));Create a list with initial value
            (if (= a 1) result;Checks when to finish loop
                ;Otherwise we check if the next value on the list is odd or even
                (if (even? a)
                    (loop (/ a 2) (append result (list (/ a 2))));If 'a' is even
                    (loop (+ (* a 3) 1) (append result (list (+ (* a 3) 1))));If 'a' is odd
                );End of fourth if
            );End of third if
        );End of second if
    );End of first if
  );end of loop
);End of first exercise

(define (hailstone-list low up);Second exercise, recieves lower and upper limit
  (let looperino;Create the first anonymous function
```

```
  ([a low] [result empty]);Start from lower limit and create a
  (if (<= a up);If the lower limit hasn't reached the upper one we run hailstone
    (looperino (+ a 1) (append result (list (hailstone a))));Create lists by using hailstone
    result;We return result
    );End of if
  );end of anonymous function
);End of Second exercise

(define (shift-char symbol amount);Recieves a char and an integer
 (if (char-alphabetic? symbol);We check if we recieved a letter
    (let loop ;Create anonymous function
     ([letter symbol] [move amount]);Assign starting values
     (if (= move 0) letter;If the amount we move is 0 we return the char
        ;Else we shift character
        (cond
          ;Cond with letter "a" and moving to the left
          [(and (= (char->integer letter) 97) (< move 0))
           (loop (integer->char 122) (+ move 1))]
          ;Cond with letter "z" and moving to the right
          [(and (= (char->integer letter) 122) (> move 0))
           (loop (integer->char 97) (- move 1))]
          ;Cond with letter "A" and moving to the left
          [(and (= (char->integer letter) 65) (< move 0))
           (loop (integer->char 90) (+ move 1))]
          ;Cond with letter "Z" and moving to the right
          [(and (= (char->integer letter) 90) (> move 0))
           (loop (integer->char 65) (- move 1))]
          ;Cond if the letter moves to the left
          [(> move 0)
```

```
            (loop (integer->char (+ (char->integer letter) 1)) (- move 1))]
           ;Cond if the letter moves to the right
           [else
            (loop (integer->char (- (char->integer letter) 1)) (+ move 1))]
           );End of condition
        );End of second if
       );End of anonymous function
     symbol;If char isn't letter do nothing
     );End of if
);End of third exercise


(define (caesar-encode string amount bool);3rd exercise, recieves string int and bool
 (let loop;Create anonymous function
   ([letters (string->list string)] [times (string-length string)];Assign starting values
                                [res empty]);Assign starting values
   (if (= times 0)(list->string res);End of loop, we return result
      ;Else we check boolean
      (if (eq? bool #f);If the boolean is false we encode the string
         (loop (cdr letters) (- times 1) (append res (list(shift-char (car letters) amount))))
         (loop (cdr letters) (- times 1) (append res (list(shift-char (car letters) (* amount -1)))))
      );End of second if
   );End of first if
 );End of anonymous function
);End of fourth exercise
```