# MCN7105: Structure and Interpretation of Computer Programs Academic Year: 2021/2022

## Exercise

1. Write a program *bubblesort* that takes a list of numbers and returns the list in sorted order. For example:

   ```
   % (bubblesort (quote (1 4 2 8 5 7)))
   (1 2 4 5 7 8)
   ```

2. Write a function, *permutations*, that takes a list as input and generates a list containing all possible permutations of the list elements. Here is a sample application:

   ```
   % (permutations (quote (1 2 3)))
   ((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
   ```

3. Define a function *maximum-of-many* that takes as argument two or more numbers and returns the maximum one. e.g.,

   ```
   (maximum-of-many 10 20 30 40) => 40
   (maximum-of-many 12 8) => 12
   ```

4. Define a function *display-all*, which is a variation of the native function display. display-all takes one or more arguments and displays them.

   ```
   (display-all "foo") => "foo"
   (display-all "foo" "bar") => "foo" "bar"
   ```

5. Define a function sum-them-all that takes one or more arguments and returns their sum e.g.,

   ```
   (sum-them-all 10 20 30 40) => 100
   (sum-them-all 12 8 2) => 22
   ```

6. Using the function *sum-them-all* defined in (5), define a function *average* that takes one or more arguments and returns their average e.g.,

```
(average 10 20 30 40) => 25
(average 10 20 30) => 20
(average 10) => 10
```

7. Consider the following function of *square-list*, which takes a list as an argument and returns a list of squares of the list elements.

```
(define (square-list lst)
(define (square x) (* x x))
(define (square-iter lst sqrd-lst)
(if (null? lst)
sqrd-lst
(square-iter (cdr lst)
(cons (square (car lst))
sqrd-lst ))))
(square-iter lst '()))
```

   (a) Unfortunately, the *square-list* function has a bug and does not work as intended. Identify the bug.

   (b) Rewrite the function *square-list* to fix the bug.

   (c) Write a recursion version of *square-list.*

8. Write a function *intermix* that takes two lists as arguments and returns a new list with the elements mixed together by alternating each element of the first list and the element of the second list. e.g.,

```
(intermix '(1 2 3 4 5 6) '(a b c d e f))
=> (1 a 2 b 3 c 4 d 5 e 6 f)
```

   Make sure that your function works for lists of different lengths.

# The End