CvP
Programming Assignment 1
Scheme

In this assignment you will practice Scheme programming. Your task is to define and test functions described below in this document.

If not stated otherwise, you are allowed to use only the built-in Scheme list operators such as CAR, CONS, CDR, etc. You can use a Scheme interpreter of your choice. For example the MIT one http://www.gnu.org/software/mit-scheme/ or the Racket Lang environment http://racket-lang.org/ (which is available in the lab machines). Refer to the respective documentations for instructions on how to use them.

Your task is to define the functions below in a single Scheme file.

APPEND

Define and test a function that appends two lists. The expected behavior is shown below.

```
> (append '(1 2 3) '(4 5 6))
(1 2 3 4 5 6)
> (append '() '(4 5))
(4 5)
> (append '(a b) '())
(a b)
> (append '(1 (2 3)) '(4 5))
(1 (2 3) 4 5)
```

INTERSECTION

Define and test a function that takes two lists and returns their intersection.

If an element appears more than once in one list or the other, it should appear in the output list only once. The expected behavior is shown below.

```
> (intersection '(1) '(1))
(1)
> (intersection '(1 2) '(1))
(1)
> (intersection '(2 1 2) '((1) 2))
(2)
```

ZIP

Define and test a function that takes two lists and pairs up corresponding elements of the two lists. If one of the lists is longer than the other, the extra elements are ignored.

The expected behavior is shown below.

```
> (zip '(1 2 3) '(a b c))
((1 a) (2 b) (3 c))
> (zip '(1 2) '(a))
((1 a))
```

FLATTEN

Define and test a function that takes a list and returns a list of all its primitive elements (numbers, atoms, etc.) and the ones of its sub lists. The expected behavior is shown below.

```
> (flatten '())
()
> (flatten '(1))
(1)
> (flatten '((1 2) 3)
(1 2 3)
```

MAP

Write a Scheme function `(map f l)` that takes two parameters, `f` and `l` and returns a list. The parameter f is a function with a single parameter, and l is a list. The returned list is the result of the application of f to the respective elements of l. For instance, if double is a function defined as

```
(DEFINE (double x) (* 2 x))
```

then

`(map double '(1 3 5 7 9))` must return the list

`(2 6 10 14 18).`

PERMUTE

Write a function in Scheme that generates all permutations for a list of *n* different elements. For example,

```
permute '(1 2 3))
```

should return

```
((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
```

Hint: Write a function in Scheme that removes a given element from a given list, construct lists composed of the item you remove from the list and the set of permutations of the list without this item. Apply the same operation to each element in the list and append all lists you obtain to construct a final list with all permutations. Use *lambda* expressions and standard Scheme functions *cons*, *append*, *map*.

SUBMISSION INSTRUCTIONS

The Scheme source file containing the function definitions should be

submitted to "Sander van Rijn" svr003@gmail.com

Remember to include your full name and student number in the body of the email.