MCN7105: Structure and Interpretation of Computer Programs

Exercise 2

1. Consider the following simple phone book directory. The entire directory is represented as a list of contacts. Each contact is made using the following constructor:

```
(define (make-contact person phone company)
   (list person phone company))
```

The person part of the contact is created using the following constructor:

```
(define make-person list)
```

Note that the values for names, phone number, and company are represented using strings. Here is an example phone book directory.

```
(define sampledata
   (list (make-contact (make-person "John" "Okot") "07982802" "UBC")
      (make-contact (make-person "Mary" "Mukasa") "09818181" "KCCA")
      (make-contact (make-person "James" "Mikisa") "0781919191" "BOU")))
```

Note that the first name is always the first element of the person abstraction, but there can be arbitrarily many last/other names.

i Complete the contacts abstraction by providing the selectors for *person*, *phone number*,and *company*. For example

```
(define entry (make-contact (make-person "John" "Okot") "07982802" "UBC"))
(person entry) -> ("John" "Okot")
(phone-number entry) -> "07982802"
(company entry) -> "UBC"
```

ii Complete the person abstraction by providing selectors for first name and other names.

```
(define sample-person (make-person "John" "Okot" "Okello"))
(first-name sample-person) -> "John"
(other-names sample-person)-> ("Okot" "Okello")
```

2. The procedure *square-list* takes a list of numbers as argument and returns a list of the squares of those numbers.

```
(square-list (list 1 2 3 4))
-> (1 4 9 16)
```

Here are two different definitions of *square-list*. Complete both of them by filling in the missing expressions.

```
(define (square-list items)
   (if (null? items)
      nil
   (cons <??> <??>)))

(define (square-list items)
   (map <??> <??>))
```

3. With a Scheme code example, explain the difference between lexical and dynamic scope in a programming language.

4. Draw an environment structure created by evaluating the following expressions

```
(define a 10)
(define (square x) (* x x))
(square a)
```

The End