Coursework Specification

Your overall task for the COMP1811 Scheme Project Coursework is to design and develop a small program in Scheme. You are expected to work individually to develop this project.

Implementing the program will give you experience in the development of an application using Functional Programming, where you are expected to design, implement, and use your own code from scratch. Your code development is the culmination of all that you have learned and all your hard work for part two of this module (Functional Programming).

Your solutions are expected to be integrated into the code template provided. Please, stick to the template given to you and complete the Scheme definitions with your own code.

It is recommended to switch off your mobile. Programming requires mindfulness.

Please read the entire coursework specification.

Scenario

You are programming your first App in Scheme: "*Restoring Facebook Social Network after it was hacked!*".

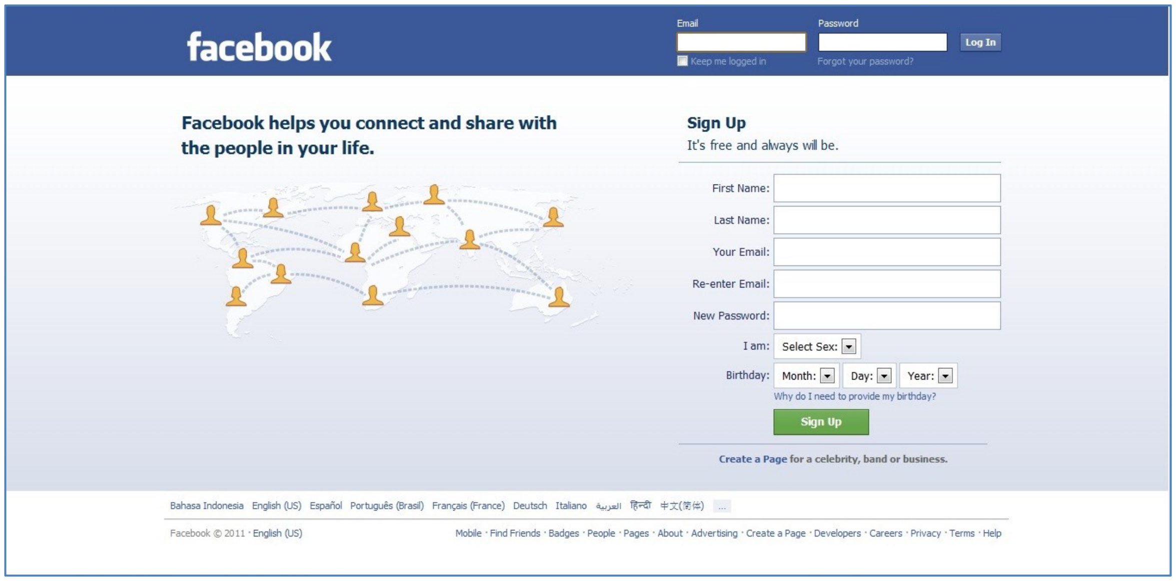

facebook
Email
Password
Log In
Keep me logged in
Forgot your password?
Facebook helps you connect and share with the people in your life.
Sign Up
It's free and always will be.
First Name:
Last Name:
Your Email:
Re-enter Email:
New Password:
I am: Select Sex:
Birthday: Month: Day: Year:
Why do I need to provide my birthday?
Sign Up
Create a Page for a celebrity, band or business.
Bahasa Indonesia English (US) Español Português (Brasil) Français (France) Deutsch Italiano العربية हिन्दी 中文(简体) ...
Facebook © 2011 · English (US)
Mobile · Find Friends · Badges · People · Pages · About · Advertising · Create a Page · Developers · Careers · Privacy · Terms · Help

Figure 1 Facebook network as a graph

Description of Coursework

Mark Zuckerberg has two reasons to worry. **Facebook**© datacentre has been hacked and apparently, the software has been partly corrupted.

Consultants have identified the damaged parts of the software and produces the **Table 1** Damage report – see Table 1 below. Luckily, the software was written in *Scheme*, using the *Functional Programming* paradigm. So, the software can be very easily repaired in isolation and restored despite the overall complexity of the system. That is one of the main principles of FP, *referential transparency*, or in Leibniz's words: *"Equals replace equals"*.

You are expected to repair individual definitions in the given code template. Let the force be with you, and *don't forget to format your code with **Ctrl+i*** !

Task 0

Familiarise yourself with the given code.

1) Acquire the source code from [sn-scheme-template.zip]() on Moodle.
2) Download, unzip, and run it in **DrRacket (**open **sn-app.rkt** and run).
3) **testing.rkt** is a test harness that runs the definitions that you will need to complete. As you implement more and more functions, the template will work as expected. See the demo video on Moodle showing the individual commands that the make use of the definitions you will complete.
4) Locate the files and the functions you are expected to implement (check **Damage report –** Table 1 below).
5) Please DO NOT alter the rest of the definitions in sn-app.rkt. You may read but not modify them.

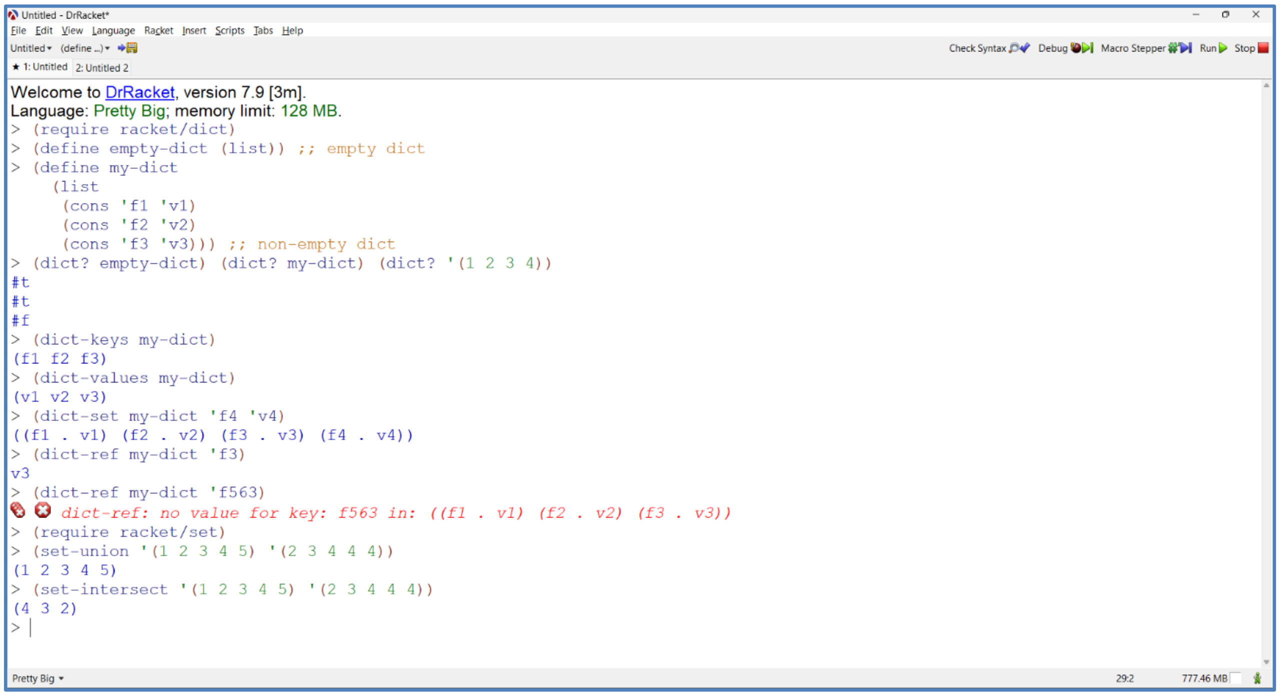
You must program using only *pure, side-effects free functions, using recursion or high-order programming.* In practical terms, you cannot use **set!, printf, read, for-each, begin** or any other side effect functions in Scheme[1]. Function with side effects conventionally have an exclamation mark (!) as the final character of the function name.

Fundamentals.

A ***social network*** - see Figure 1, can be viewed in mathematical terms as a **graph** (network), made of **nodes** (users) and **edges** (links) among them. Essentially you can:

- add users (**nodes**)
- connect friends (link **nodes** by **edges**)
- count the number of users (count how many **nodes**)
- count the number of friends a user has (count how many **edges** a **node** has**)**
- computing the friendliest (unfriendliest) user (**node** having max(min) number of **edges**).
- others…

It may be implemented using **lists or dictionaries.** In **Scheme,** everything is a **list**, so a **dictionary** is nothing but a **list of pairs entry-value**.


Untitled - DrRacket*
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...)
Check Syntax Debug Macro Stepper Run Stop
1: Untitled 2: Untitled 2
Pretty Big
29:2
777.46 MB

```
Welcome to DrRacket, version 7.9 [3m].
Language: Pretty Big; memory limit: 128 MB.
> (require racket/dict)
> (define empty-dict (list)) ;; empty dict
> (define my-dict
    (list
     (cons 'f1 'v1)
     (cons 'f2 'v2)
     (cons 'f3 'v3))) ;; non-empty dict
> (dict? empty-dict) (dict? my-dict) (dict? '(1 2 3 4))
#t
#t
#f
> (dict-keys my-dict)
(f1 f2 f3)
> (dict-values my-dict)
(v1 v2 v3)
> (dict-set my-dict 'f4 'v4)
((f1 . v1) (f2 . v2) (f3 . v3) (f4 . v4))
> (dict-ref my-dict 'f3)
v3
> (dict-ref my-dict 'f563)
dict-ref: no value for key: f563 in: ((f1 . v1) (f2 . v2) (f3 . v3))
> (require racket/set)
> (set-union '(1 2 3 4 5) '(2 3 4 4 4))
(1 2 3 4 5)
> (set-intersect '(1 2 3 4 5) '(2 3 4 4 4))
(4 3 2)
> |
```

RATIONALE: A **node** can be viewed as the **key**, and the list of friends as the **value** for that key[2].

[1] You may use it for debug purposes, dropping them in final release.
[2] Note dictionaries are useful in finding the *intersection and unions between lists.*

Table 1 Damage report

MODULE	FUNCTION
sn-app.rkt (ROM)	Undamaged – no changes required. **Do not alter**.
sn-console.rkt (ROM)	Undamaged – no changes required. **Do not alter**.
sn-io.rkt (ROM)	Undamaged – no changes required. Undamaged – no changes required. Do not alter.
sn-graph.rkt	(Contains damaged functions)
sn-social-network.rkt	(Contains damaged functions)
sn-utils.rkt	(Contains damaged functions)

Task 1

The following are the specifications of the functions that need to be *repaired*. Be aware that functions can use existing functions, i.e., they can call other functions that have been defined or found from libraries.

The following are the damaged files for you to correct:-

File 1 - sn-utils.rkt (Damaged)

i. sn-list->dict (1 mark)

Given a list of entries, this function should return the argument passed. This function creates a list of pairs.

```
> (sn-list->dict (list (cons 'k1 'v1) (cons 'k2 'v2)))

((k1 . v1) (k2 . v2))
```

ii. sn-dict-ks-vs (2 marks)

Given a **list** of **values** and a list of user IDs/Names (**keys)**, this function should return a **dictionary** (remember - a list of keys with their corresponding values).

```
> (sn-dict-ks-vs '(k1 k2 k3) '(v1 v2 v3))

((k1 . v1) (k2 . v2) (k3 . v3))
```

iii. sn-line->entry (4 marks)

Given a **string**, this function should first split it into a list, then return a **dictionary entry,** i.e **a list** where the first element is the **key** and the remainder of the list is the corresponding **value** as **symbols**[3].

```
> (sn-line->entry "A B C D")

(A B C D)
```

[3] The following procedures: **string-split** and s**tring->symbol** may be used here**.** You will need to add **(require racket/string)** to your code to import those procedures.

File 2 - sn-graph.rkt (Damaged)

The template given to you on Moodle, **sn-scheme-template.zi[,** contains **testing.rkt** which can be used to test all your functions.

```
> (define my-dict
 (list
  (cons 'f2
      (list 'f3 'f4))
  (cons 'f3
      (list 'f2))
  (cons 'f4
      (list 'f3 'f2))
  (cons 'f13
      (list ))
  (cons 'f1
      (list ))
  ))
```

```
 > my-dict
((f2 f3 f4) (f3 f2) (f4 f3 f2) (f13) (f1))
```

i. sn-empty (1 mark)

Define the constant named *sn-empty* as the empty list. *sn-empty* defines the **empty** network.

```
> sn-empty

()
```

ii. sn-users (2 marks)

This function returns the **list of users** (IDs or names) in the given network, *my-dict*.

```
> (sn-users my-dict)

(f2 f3 f4 f13 f1)
```

iii. sn-add-user (5 marks)

Given a network and an ID or name, this function should add a new entry to the network for that user. Before you add the user you should check whether that user already exists and if they do return the network unchanged.

```
> (sn-add-user my-dict 'f5)

((f2 f3 f4) (f3 f2) (f4 f3 f2) (f13) (f1) (f5))
```

iv. sn-add-frndshp (5 marks)

Given a network and two existing **keys** (user IDs or names), this function should update both users' list of friends.

```
> (sn-add-frndshp my-dict 'f1 'f2)

((f2 f1 f3 f4) (f3 f2) (f4 f3 f2) (f13) (f1 f2))
```

File 3 - sn-social-network.rkt (Damaged)

i. sn-ff-for (2 marks)

Given a **network** and an existing ID/name, this function should return the list of friends for that user ID/name.

```
> (sn-ff-for my-dict 'f4)

(f3 f2)
```

ii. sn-cmn-frnds-btwn (6 marks)

Given a network and two existing IDs/names, this function returns the list of friends of friends in common between these two users.

```
> (sn-cmn-frnds-btwn my-dict 'f2 'f4)

(f3)
```

iii. sn-frnd-cnt (4 marks)

Given a network, this function returns a list of pairs. Each pair contains a user ID/name and the number of their friends.

```
> (sn-frnd-cnt my-dict)

((f2 . 2) (f3 . 1) (f4 . 2) (f13 . 0) (f1 . 0))
```

iv. sn-frndlst-user (4 marks)

Given a non-empty network, this function returns a pair consisting of the user ID/name of the person having the maximum number of friends in the network and the number of friends that they have.

```
> (sn-frndlst-user my-dict)

(f2 . 2)
```

v. sn-unfrndlst-user (4 marks)

The same as previous, but with minimum number of friends.

```
> (sn-unfrndlst-user my-dict)

(f1 . 0)
```

Task 2

Write a **brief report** describing the main FP topics (one statement in plain English, please) you used to implement each function (i.e., *recursion, high-order, lists, union, intersection, dictionaries, meta-programming…)*.

Upload the folder containing the **Scheme code you developed (.rkt files), your final report as a pdf file, and a 5–10-minute screencast (explaining the code and showing your system running)** to Moodle under the Scheme Project Upload link under "CW Details and Submission" **by 23:30 on 20/03/2023**.

You are also required to **demo your work** during your scheduled lab session in the week commencing 20/03/23 in agreement with your lab tutor. You may fail this assessment if you fail to attend the demonstration without extenuating circumstances.