CSI 3350 PROGRAMMING LANGUAGES
- Fall 2019 Homework 01 -

Guidelines:
This file should be accompanied by three other source files:
  hw01-answer-sheet.rkt
  hw01-tests.rkt
  test-infrastructure.rkt

DUE: Friday, Sep 20, 2019 at 11:55pm

All of the above files will have to be in the same folder for the tests
to work. The questions that require you to write code are accompanied by test cases. You can see in the description of the test cases for which
problems they are written. To run the tests check the "hw01-tests.rkt" file for further instructions.

The tests are written using the small library defined in "test-infrastructure.rkt", check that file for further details about the testing infrastructure.
--
Submission guidelines:
  - first, write your answers in the accompanying answer sheet file
    named "hw01-answer-sheet.rkt".
  - then, rename the answer sheet file to "hw01-yourlastname.rkt" and
    (replace "yourlastname" with your real last name)
  - upload/submit ONLY the renamed answer sheet file.

Possible penalties:
  - 5% of the total points for not renaming the answer sheet
  - up to 10% for submitting a file that does not compile (to compile,
    hit the "Run" button from the DrRacket UI when your file is open).
    See the answer sheet for further details.
  - 25% penalty for submissions after Friday (Sep 20 @ 11:55pm) but
    before Sunday (Sep 22 @ 11:55pm).
  - Your submission will NOT be accepted after Sunday(Sep 22)@11:55pm
==

--
In Racket/Scheme, all expressions are written between a pair of parentheses.
Whenever the interpreter encounters an open parenthesis, e.g.:
(+ 3 4)
it will expect that the first element of the list to be a function (except when using the ' (quote) function); in Scheme operators are also

functions (we will cover this in more detail next week).

The implication of this uniform way of treating things is that mathematical
expression have to be written in prefix notation (the operator comes first, the operands later) as opposed to the infix notation we see in Java. But, this allows for addition to take an arbitrary number of operands:
>(+ 1 2 3 4)
10

The ">" symbol is the prompt: the indication that the expression following
it is evaluated in the interpreter.
==========
1. (complete the following, we did similar calculation in class)

Translate the following algebraic formulas into Scheme's notation. Type the translation into Scheme's interaction window for checking.
((3 + 3) ∗ 9)
((6 ∗ 9) / ((4 + 2) + (4 ∗ 3)))
(2∗ ((20 − (91 / 7)) ∗ (45 − 42)))
==========
2.

Describe in words how to translate an algebraic formula into Scheme's notation.Be sure to handle the general case.
==========
3.

Using Scheme's define special form, write definitions of variables x, y and z,such that x has a value of 2 and y has a value of 3 and z has a value of 4.
==========
4.

Write and evaluate an if (or cond) expression in Scheme that uses the variables x, y and z, and that has as its value the sum of the two largest variables, and, if all are of the same value, the value of the function is 0 (zero).

==========
5.

Evaluate a similar if (or cond) expression that has as its value the sum of the two smallest of x, y and z, if all three are of the same value, the value of the function is 0 (zero).
==========
6.

Without using an if (or cond) expression, write an expression that uses both variables x and y, which has as its value #t if the values of x and y are equal, and #f otherwise.

==========
7.
What is de difference between the following two statements? State your answer in English.

(define thirty-five 35)
(define (thirty-five) 35)
==========

We will cover list data type in more details as we proceed, but let's first mention it before hand:

Scheme offers a "one size fits all collection", list. It is a homogeneous collection,i.e. it can store any number of types of values at the same time. Very important for you to remember is that lists are immutable; all operations on lists (car, cdr cons) do not modify the original list, but rather construct a new one. We will study the implications this fact has on our programming style in later homework. For now, we will focus only on constructing lists.

==========
==========
8.

Experiment with ' (quote) in the Scheme interpreter. Try, for example 'name, '+,
'(/ 4 2), 'gargleblaster, and unquoted versions of expressions.
In English, give a precise answer the following question: what does ' do?
==========
==========
9.

A similar procedure to ' (quote) is list. What is the difference between ' and list?
Hint: try using variables and function calls in conjunction with the two.
==========
==========
10.

What can you do with a string in Scheme that you can't do with a symbol?
Why is there a distinction between strings and symbols in Scheme?
(Hint: look at the Revised Report on Scheme, url: http://www.r6rs.org/, to see the operations that Scheme defines for these

types.)
==========
11.

Using *only* the Scheme procedure 'list' and numbers and quoted
symbols (such as '*, 'name, and 'james),
write Scheme expressions to make the following lists.

(Note that the line breaks and indentation in the last of these lists
do not matter;
the interpreter will print out the value of your answer without this
indentation,
which is OK. We are asking for you to create the values displayed
below, not the
printing displayed below. So do NOT use \newline or #\space in your
answer.)

(4 2 6 9)

```
(spaceship
       (name(serenity))
       (class(firefly)))
```

(2 * ((20 - (91 / 7)) * (45 - 42)))
==========
12.
Suppose we are writing code for a procedure in which the variable
"lst" is bound to a
list of numbers. Suppose further that lst has the value:

    (a b c)
For each of the following, write an expression that uses lst and makes
the given list:

    (d a b c)
    (a b d a b)
    (b c d a)

Hint:
use the functions cons, car and cdr or other list accessor shorthands
found in the following
url:
http://docs.racket-lang.org/reference/pairs.html?
q=cadr&q=rackunit#(part._.Pair_.Accessor_.Shorthands)
==========
13.

The standard library provides two functions eq? and equal? (the question mark
is part of the name). Experiment with these two functions and concisely describe
the difference between them.
==========
14.

Using string manipulation function like: string-append, number->string and symbol->string to:
Define the function create-error-msg that behaves in the following manner:

(create-error-msg 'answer-to-everything 42)
"This is a custom error message we will be using next. Symbol 'answer-to-everything was not paired with value 42"

You may assume that the first argument is always a symbol and that the second one is
always a number.
==========
15.

For testing purposes we will use exceptions instead of errors (as you might notice
from your textbook). We will not go into detail of how to handle them; all you need to know is that you
"raise" an exception with the following statement:

> (raise 42)
uncaught exception: 42

> (raise '(42 42))
uncaught exception: '(42 42)

> (raise "forty two")
uncaught exception: "forty two"

Where the first argument can be any Scheme value: number, list, string, to name a few. In later homeworks,
you will be asked to write functions that result in specific error messages and we will use the value
of the parameter to automatically check the intended semantics.

----DO THE FOLLOWING---
Write a function, i.e., check-correctness, that takes a 2-element list, where the 1st element is a symbol and
the 2nd is a number. (From here on, a 2-element list is referred to as

a pair or 2-tuple) .

The following contract should be enforced by your function: if the symbol is 'answer-to-everything but the
number is not 42, then raise an exception with the error message that you create with the function defined
for the previous question; your function returns #t only when the pair is exactly '(answer-to-everything 42);
for all other cases your function should return #f.

```
>(check-correctness '(answer-to-everything 42))
#t

>(check-correctness '(symbol-other-than-the-previous-one 42))
#f

>(check-correctness '(test 30))
#f

>(check-correctness '(answer-to-everything 10))
uncaught exception: "This is a custom error message we will be using next. Symbol
'answer-to-everything was not paired with value 42"
```

==========

16.

A fast and useful debugging tool is printing out relevant information. Two
functions you will encounter during this course are:

```
>(display 42)
42

>(print 42)
42
```

The semantic difference between the two is not relevant for this course. You are,
however, encouraged to look into this matter by yourself.

To print a newline evaluate the function:
```
>(newline)
```

Because the branches of an "if" instruction can contain only one instruction each,
in order to print any relevant information from any branch and still

do computation
you can use the "begin" function:

```
>(if true
      (begin (print "computing 42") (newline) (+ 12 30))
      (print "dead code")
   )
```

A begin function can contain 0 or more instructions. The value of the begin
function is always the value of the last instruction. As you can see from
the above example, the first two instructions produce side effects (printing)
while the last instruction computes a value which is then returned as the
value of the "begin" instruction and then taken as the value of the "if"
instruction.

You do not have to do anything for this problem, just experiment it a bit.

==========