

This assignment is to get you started programming in Scheme. You shouldn't have any trouble with the algorithmic content of these problems, although there will be some adjustment if you are not accustomed to functional programming.

Here are the standard Scheme functions we will have mostly covered in the first week or so:

- variable/function definition: `define`, `let`, `let*`
- list functions: `length`, `car`, `cdr`, `list-ref`, `list-tail`, `cons`, `list`, `append`, `reverse`, and the `car/cdr` combinations
- conditionals: `if`, `cond`, `case`
- mathematical predicates: `=`, `<`, `>`, `<=`, `>=`, `zero?`, `positive?`, `negative?`, `even?`, `odd?`, `number?`, `real?`, `integer?`, `complex?`, `rational?`
- non-mathematical predicates: `equal?`, `list?`, `null?`, `symbol?`
- boolean operators: `and`, `or`, `not`
- mathematical functions: `+`, `-`, `*`, `/`, `quotient`, `max`, `min`, `truncate`, `round`, `floor`, `ceiling`, `sqrt`, `abs`, `remainder`, `modulo`, `gcd`, `lcm`, `expt`, `exp`, `log`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

This assignment can be done only using the above functions, but you may also use MIT Scheme extensions such as `sublist`. I encourage you to stick to standard Scheme as much as possible. You MAY NOT use "advanced" features of Scheme at this point, in particular: `map`, `apply`, `assoc`, `set!`, iterative functions, and any of their variants.

You are to turn in your answers for this assignment on paper. If the automated code testing is working before the assignment is due, we may ask that you upload your code to test the system.

1. (3 points) Assume that we will represent a date by a list of three elements: `(month day year)`, where `month` is one of the following symbols: `jan`, `feb`, `mar`, `apr`, `may`, `jun`, `jul`, `aug`, `sep`, `oct`, `nov`, `dec`.

Write functions `(month d)`, `(day d)`, and `(year d)` which return the proper component of the date `d`. For example:

```
(define bday '(jan 12 1997))
(month bday)      ==> jan
(day bday)       ==> 12
(year bday)      ==> 1997
```

These sorts of accessor functions will be useful, particularly when we start encoding more complex information into a list data structure.

2. (6 points) Under the Gregorian calendar, a year is a leap year if either:

- the year is divisible by 4 but not by 100
- the year is divisible by 400

Write a predicate `(leap-year? year)` that determines whether `year` is a leap year. Note that a predicate should return either `#t` or `#f` (and in MIT Scheme, `#f` is equivalent to `()`).

3. (6 points) Write a function `(days-in-month month year)` that returns the number of days in the given month. Assume `month` is one of the three letter abbreviations as in Problem 1.

4. (6 points) Write a function `(day-of-year date)` which returns the day of the year of the given date, where `date` is specified as in Problem 1.

You may not use any numbers in your solution! Your function (or its helper function) should recursively add up the number of days in the preceding months. You should use the functions you've written for the previous questions. You will find it helpful to use the following definition:

```
(define months '(jan feb mar apr may jun jul aug sep oct nov dec))
```

Here are some examples:

```
(day-of-year '(jan 31 2000))    ==> 31
(day-of-year '(feb 1 2000))    ==> 32
(day-of-year '(sep 8 2000))    ==> 252
```

5. (5 points) Suppose we make the following definition:

```
(define m '((a b) (c (d e)) f ((g))))
```

Write expressions using *only* the `car` and `cdr` functions (and their abbreviated combinations such as `cadadr`) which operate on the variable `m` to produce the following values:

- (a) `c`
- (b) `(f ((g)))`
- (c) `((d e))`
- (d) `b`
- (e) `g`

For example, `(caar m)` or `(car (car m))` will return the value `a`.

6. (6 points) Assume that the following definitions have been made:

```
(define colors '(red blue))
(define primes '(7 11 13 17))
(define deep-list '((#t)))
(define sign 'stop)
```

Using only these four variables and the functions `cons`, `list`, and `append`, write expressions that will return the following lists:

```
(stop 7 11 13 17)
(7 11 13 17 stop)
((red blue) (7 11 13 17) (((#t))))
(((#t)) 7 11 13 17 red blue)
((red blue) (7 11 13 17) ((#t)))
(((7 11 13 17)) ((#t)) (red blue))
```

7. (6 points) Write the function `(flatten lst)` which takes a list and returns a "flattened" version of it, i.e. without any sublists. For example:

```
(flatten '(1 (2 wombat (5)) (6 7))) ==> (1 2 wombat 5 6 7)
```

8. (6 points; hard!) Write a function `(make-list n)` which returns a list of length `n` where the i^{th} element is a list of the integers from 1 to i . For example:

```
(make-list 4) ==> ((1) (1 2) (1 2 3) (1 2 3 4))
```