These are some common mistakes on Midterm 2, we hope that this can help you with future assignments and exams.

**Q1: Zaphod**

**Part(a): writing a Zaphod**
- Not producing the answer with as few other values as possible
  (e.g. should be 42 but students wrote `(make-zaphod 42 true)`)  

**Part(b): template**
- Missing conditions for boolean, integer and string
  
  (e.g. missing  
  ```  
  [(boolean? zaphod) ...]  
  [(integer? zaphod) ...]  
  [(string? zaphod) ...]  
  ```)
- Missing recursive call
  (e.g. should be  
  ```
  (...(zaphod-template (zaphod-beeble zaphod))...  
  ...(zaphod-template (zaphod-brox zaphod))...)
  ```

**Q2: Stepping Question**

**Part(a)**
- Substituting in 5 and 10 for the second step
  (e.g. Writing 5 and 10 when it should be a and b)

**Part(b)**
- Incorrect final step
  (e.g. Writing `(cons 5 empty)` when it should be `(cons 5 (cons empty empty))`)

**Part(c)**
- Generally well-done
Part(d)
- Next step of `(rest '((1 2) (1 2)))` is `(list (list 1 2))`, not `(list 1 2)`

Part(e)
- Not realizing that constants are evaluated immediately when they are defined
  (e.g. Next step of `(append s0 s1)` is `(append (list 2 4 6 8) s1)`),
  not `(append (silly '1 2 3 4 5 6 7 8) 0) s1)`

Q3: Working with List

Part(e)
- Omitting one of the empty

Part(g)
- Writing `(cons 1...` when it should be `(cons (first q1)...`
- Writing `(cons (rest list)...` when it should be `(cons (rest q1)...`

Part(h)
- Generally well-done

Q4: De-bounce
- Missing `(cons (first lst)` before taking the rest in the recursive call, which fails to preserve the elements in the list
- Incorrect recursion involving `(second lst)` or `(rest (rest lst)` when it should be `(rest lst)`

Q5: Insert List into List
- Not using the main function as a wrapper, which complicates the code
- Writing `else (cond ...`, which could be simplified

`insert-at`
- Missing `list->string` at the beginning
- Not using `string->list` on `s1` and `s2` which complicates `insert-at/list`. We accepted such solutions so long as they produced the right types, but getting this correct was pretty difficult
insert-at/list

- Wrong syntax for list in contract
  (e.g. should be (listof Char) but students wrote List)
- Missing or wrong requirements
  (e.g. should be n <= (length loc1) if the parameter consumed is a list)
- Using cons like append
  (e.g. (cons loc2 loc1) but cons can only be used to add one character at a time)

Q6: Insert List into List

- Writing [else (cond .... which could be simplified

part(a): filter-position

- Missing one of the requirements
  (e.g. VCList is non-empty and all members are for the same electoral position)
- Incorrect return type in the base case
- Failure to use a helper function in the pure structural and accumulative solutions
- Using list or append instead of cons in recursive call

part(b): filter-position

- Forgetting the base case
- Writing (list elem (filter-position ... ))
  when it should be (cons elem (filter-position ... ))
- Incorrect or generalized comparison operator was used
  (e.g. Writing equal? or string=? when it should be symbol=?)
- Omitting position during the recursive call

part(c): combine

- Not inserting but only comparing consecutive elements, which will only work on sorted list

part(d): fill in the blanks

- Leaving off the vcount-name structure accessor to get the name from the structure produced by everything else
- Throwing in unnecessary "list" or "first" or "rest"
- Stretching filter-position to fill two slots