CAS CS 320
Lec 8

X2 & Z2

1. Higher order function Review
   a. int1_forall(n, fn i =>
      ```
                              if i < 2 then true
                              else (n mod i <>0))
      (* int1_forall to test if the value n is prime or not *)
      ```
2. "Cross product"
   a. Ex) xs = [1,2]
      ys = [a,b,c]
      cross(xs, ys) = [(1,a), (1,b), (1,c), (2,a), (2,b), (2,c)]
3. X2 and X3
   a. ```
      fun list_cross2(xs: ‘a list,ys: ‘b list): (‘a * ‘b) list =
              list_map(xs, fn x => list_map(ys, fn y => (x,y)))
              (* for each y, encounter the pair (x,y) *)
              (* when we see x, we traverse the map and when we see y, get pair (x,y) *)
              (* the output we get is (‘a * ‘b) list list so we need to flatten it *)
              (* [[(1,a), (1,b), (1,c)], [(2,a), (2,b), (2,c)]] → need to delete inner list *)
      ```
   b. Therefore, need to do
      ```
      fun list_cross2(xs: ‘a list,ys: ‘b list): (‘a * ‘b) list =
              list_concat(list_map(xs, fn x => list_map(ys, fn y => (x,y))))
      ```
   c. ```
      fun list_enumerate(xs: ‘a list): (int * ‘a) list =
              list_reverse(
                      #2(list_foldl(xs,(0, nil), fn(i,r), x) => (i + 1, (i,x) :: r)))
              )
      ```
   d. ```
      fun list_concat(xss: ‘a list list): ‘a list =
              list_foldr(xss, [], fn(xs, res) => list_append(xs,res))
              OR
              case xss of
              nil => nil
              | xss1: xss => list_append(xss1, list_concat(xss))
      ```
   e. ```
      fun list_z2map(xs: ‘a list, ys: ‘b list, fopr: ‘a * ‘b -> ‘c): ‘c list =
              list_map(list_zip2(xs,ys), fopr)
      ```
   f. ```
      fun list_x2map(xs: ‘a list, ys: ‘b list, fopr: ‘a * ‘b -> ‘c): ‘c list =
              list_map(list_cross2(xs,ys), fopr)
      ```
   g. ```
      fun list_filter(xs: ‘a list, test: ‘a -> ‘bool): ‘a list =
              list_foldr(xs,[],fn(x1, res) => if test(x1) then x1 :: res else res)
      ```

h. fun list_z2forall(xs: ‘a list, ys: ‘b list, test: ‘a * ‘b -> bool): bool =
```
        list_forall(list_zip2(xs,ys), test)
```
i. fun list_x2forall(xs: ‘a list, ys: ‘b list, test: ‘a * ‘b -> bool): bool =
```
        list_forall(list_cross2(xs,ys), test)
```
j. fun list_x2exists(xs: ‘a list, ys: ‘b list, test : ‘a * ‘b -> bool): bool =
```
        list_exists(list_cross2(xs,ys),test)
```
4. Python library
   a. def int1_forall(n0, test_func):
```
        i0 = 0
        while(i0 < n0):
                if not test_func(i0):
                        return False
                i0 = (i0 + 1)
        return True
```
   b. def int1_foreach(n0, work_func):
```
        i0 = 0
        while(i0 < n0):
                work_func(i0)
                i0 = i0 + 1
        return None
```
   c. def int1_rforeach(n0, work_func):
```
        i0 = 0
        while(i0 < n0):
                work_func(n0-1-i0)
                i0 = i0 + 1
        return None
```
   d. def int1_map_fnlist(xs, fopr_func):
```
        return foreach_to_map_fnlist(int1_foreach)(xs, fopr_func)
```
   e. def int1_map_pylist(xs, fopr_func):
```
        return foreach_to_map_pylist(int1_foreach)(xs, fopr_func)
```
   f. def foreach_to_map_pylist(foreach):
```
        def map_pylist(xs, fopr_func):
                res = []
                def work_func(x0):
                        nonlocal res # want to access variable that is outside
                        res.append(fopr_func(x0))
                        return None
                foreach(xs, work_func)
                return res
        return map_pylist
```

g. def int1_foldleft(xs,r0, fopr_func):

```
        return foreach_to_foldleft(int1_foreach)(xs, r0, fopr_func)
```

h. def in1_foldright(xs, r0, fopr_func):

```
        return rforeach_to_foldright(int1_foreach)(xs, r0, fopr_func)
```

i.