John Pan
Ex. 1

```
Mapper1:
For each input key/value pair (name, (flist))        # Use friend as key and person as value
    For each person f of flist
        Output key-value pair (f, name)

Reducer1:
count = 0
For each input tuple (f, namelist)    # if namelist does not contain both X and Y
    For each n in namelist            # then f is not a mutual friend
        count++                       # Only output tuples that contain mutual friend
    if count == 2
        Output (f, namelist)

Mapper2:
For each input tuple (f, namelist)    # Inverse the key and value
    Output (namelist, f)              # so that the shuffle step groups mutual friends in list

Reducer2:
For each input tuple ((name1, name2), flist)         # Identity function
    Output ((name1, name2), flist)                   # Note namelist = (name1, name2)
```

Example:

Mapper1:

Input:

⇨ (Joe, (Abe, Jane, Ali, Zack)) & (Ali, (Sheila, Jane, Zack, Mary, Joe))

Output:

⇨ (Abe, Joe), (Jane, Joe), (Ali, Joe), (Zack, Joe)

⇨ (Joe, Ali), (Jane, Ali), (Zack, Ali), (Mary, Ali), (Sheila, Ali)

Reducer1:

Input:

⇨ (Jane, (Joe, Ali)); (Zack, (Joe, Ali)); (Abe, Joe); (Ali, Joe); (Joe, Ali); (Zack, Ali); (Mary, Ali)

Output:

⇨ (Jane, (Joe, Ali))

⇨ (Zack, (Joe, Ali))

Mapper2:

Input:

⇨ (Jane, (Joe, Ali))

⇨ (Zack, (Joe, Ali))

Output:

⇨ ((Joe, Ali), Jane)

⇨ ((Joe,Ali), Zack)

Reducer2 :

Input/Output (identity function) :

⇨ ((Joe, Ali), (Jane, Zack))

Ex. 2

Mapper1:

For each tuple (Hname, Province) of H, output (Hname, (H, (Province)))

For each tuple (HInsurNum, age, Hname) of P for which age > 60, output (Hname, (P, (HInsurNum)))

# comments: filter on patients whose age > 60. Set key as Hname to join on.

Reducer1:

```
For each tuple (H, value-list)
    Ht = Pt = empty;
    For each v = (rel, val) in value-list
        If v.rel = H
            Insert v.val into Ht
        Else insert v.val into Pt
    For v1 in Ht
        For v2 in Qt
            Output(v1, v2)
```

# This join will output tuples where province is key and HInsurNum is value.

Mapper2:

For each tuple (p,h) output (p,h)

# identity function, shuffle step will group all HInsurNums to their respective province key.

Reducer2:

```
For each (p, hinlist) perform COUNT aggregation on hinlist          # Count
If count in (p, count) > 100                        # Output provinces with >100 patients
    Output (p, count)
```

# Count the number of persons in the value list (hinlist) and output if > 100.

Example input: (Same process for more patients. Example done on a total of 4 patients for simplicity)

(hosp1, PEI)
(hosp2, QC)
(hosp3, ON)
(hin1, 63, hosp1)

(hin2, 66, hosp1)

(hin3, 70, hosp2)

(hin4, 80, hosp1)

Mapper1:

Output:

- (hosp1, (H, (PEI))), (hosp2, (H, (QC))), (hosp3, (H, (ON)))
- (hosp1, (P, (HIN1))), (hosp1, (P, (HIN2))), (hosp2, (P, (HIN3))), (hosp1, (P, (HIN4)))

Reducer1:

Input:

- (hosp1, ((H, (PEI)), (P, (HIN1)), (P, (HIN2)), (P, (HIN4))))
- (hosp2, ((H, (QC)), (P, (HIN3))))
- (hosp3, (H, (ON)))

Output:

- (PEI, HIN1), (PEI, HIN2), (PEI, HIN4)
- (QC, HIN3)

Mapper2 (identity function):

Output:

- (PEI, HIN1), (PEI, HIN2), (PEI, HIN4)
- (QC, HIN3)

Reducer2:

Input:

- (PEI, (HIN1, HIN2, HIN4))
- (QC, HIN3)

Intermediary step for visualization:

- (PEI, 3)
- (QC, 1)

Output:

- No output since count < 100

Ex. 5

```
DESCRIBE Grpp
Grpp: {
    group: chararray,
    proj: {
        (
            prname: chararray,
            newdeaths: int
        )
    }
}
```

Ex. 6

```
DESCRIBE jndQc
jndQc: {
    QcDeaths::prov: chararray,
    QcDeaths::totalDeaths: long,
    proj::prname: chararray,
    proj::idate: chararray,
    proj::newdeaths: int
}
```