Part 1

No need for generics here, we have functions of Double to Double only

- There is set of functions that do simple calculations:
  ```
  def msq(x:Double) = { print("squaring"); x*x}
  def mcube(x:Double) = {print ("cubing"); x*x*x}
  def m.. (define two more as you like)
  ```
  These functions are not pure because produce side effect (printout in the terminal).

- **Modify** these functions to be **pure functions returning** objects of class containing the **value & the message** (call it **DocumentedValue**). Such functions are pure but not easily compassable.

- The goal is to **write monadic composition** for them. i.e. we want to be able to write:
  ```
  val result = msq.compose(mcube) (2.1)
  ```
  and the result would be DocumetedValue from where the value can be taken and the **concatenated messages** from all of the functions called.

- 6 points will be given if you manage to make the composition easier syntactically:
  ```
  val composed = myq =>> mycube =>> msth
  val result = composed(1.2);
  ```

Part 2

The goal of the exercise is to use scala collections in order to perform various tasks.
We have a list of people:

```
val people = List( ("John", "Doe", 23, "male"), ("Joan", "Doe", 23, "female"),
            ("Steve", "Jenkins", 22, "male"), ("Eva", "Robinson", 25, “female"),
            ("John", "Who", 22, "male"), ("Janet", "Reed", 21, "female"),
            ("Rob", "Jenkins", 22, "male"), ("Ella", "Dawson", 27, "female") )
```

The meaning of fields is obvious.

Using collections API **(simple for loop solutions are excluded)** obtain & print the following information.

- Obtain list of names
- Obtain set of names
- Obtain set of people age values
- Split the collection into people who are older and younger than 23
- Group people by age
- Obtain list sorted by age
- Check if among the people on the list there are people of names Joe or Rob.
- In a **single** set of transformations check if the number of males and females in the list is identical.