Functional Programming Exercises

Ex1: Sum

1. write function `calcSum2` which takes 2 integers and returns the sum.

```
def calcSum2(a:Int,b:Int):Int
```

2. write function `calcSum3` which takes 3 integers and returns the sum.

```
def calcSum3(a:Int,b:Int,c:Int):Int
```

3. write function `calcSumList` which takes list of integers and returns the sum.

```
def calcSumList(a:List[Int]):Int
```

Note:

- compare Pure/Impure function.
- re-use function.
- some of this can be `Recursion` (Recursive)

Ex2: Max

1. write function `findMax2` which takes 2 integers and returns the greater number.

```
def findMax2(a:Int,b:Int):Int
```

2. write function `findMax3` which takes 3 integers and returns the max number.

```
def findMax3(a:Int,b:Int,c:Int):Int
```

3. write function `findMaxList` which takes list of integers and returns the max number.

```
def findMaxList(a:List[Int]):Int
```

Note:

- maybe we can build sort function

Ex3: Higher Order Functions (Function as Parameter)

1. write function `plus` which takes 2 integers and returns the result.

```
def plus(a:Int,b:Int):Int
```

2. write function `minus` which takes 2 integers and returns the result.

```
def minus(a:Int,b:Int):Int
```

3. write function `multiply` which takes 2 integers and returns the result.

```
def multiply(a:Int,b:Int):Int
```

4. write function `calc` which takes 2 integers and 1 function which takes 2 integers and returns the result. //try to pass C1,C2,C3 into C4

```
def calc(a:Int,b:Int, c: (Int,Int) => Int)
```

5. write function `getCalcMode` which takes 1 Mode and return the related function //hint: pattern matching

```
case object ModePlus extends Mode
case object ModeMinus extends Mode
case object ModeMultiply extends Mode
```

```
def getCalcMode(mode: Mode):  (Int, Int) => Int
```

6. write function `calcByMode` which takes 2 integers and 1 Mode and returns the result.

```
def calcByMode(a: Int, b: Int, mode: Mode): Int
```

Ex4: Map

1. write function `multiplyTwo` which takes 1 integers (a) and returns the result (a*2).

```
def multiplyTwo(a:Int):Int
```

2. write function `calcList` which takes list of int and 1 function which takes 1 integers and returns the list that already applied the function. //use D1 as param

```
def calcList(a:List[Int], c:Int => Int)
```

Note: Just like a .map

Ex5: Option DataType

1. Write function `find` which takes list of User and a target id and return result.

//use Option

```scala
    case class User(id: Int, name: String)
    def findUserById(a:List[User], id: Int):Option[User]
```