Functional Programming

To begin:

- In Eclipse create a Scala project called FunctionLab.
- Add a worksheet to this project called session
- Add a Scala interpreter to this project

In the session worksheet define and test the following functions. Your implementations should be as concise as possible.

To end:

- Export session.sc to your file system and send it to the grader by the deadline.

1. Problem

Recall that the composition of two functions—f and g-- is h(x) = f(g(x)). Of course the range of g must be a subset of the domain of f. Implement a generic compose combinator.

2. Problem

Use recursion and your solution to problem 1 to implement the self-composition iterator combinator:

```
def selfIter[T](f: T=>T, n: Int) = f composed with itself n times.
```

Test your function with:

```
def inc(x: Double) = x + 1
def double(x: Double) = 2 * x
```

Note:

```
selfIter(f, 0)= id where id(x) = x.
```

3. Problem

Write a function called countPass that counts the number of elements in an array of elements of type T that pass a test of type T=>Boolean.

4. Problem

A. Most traditional-style recursive functions have similar implementations. Formalize this similarity by implementing a recur combinator that takes two inputs:

```
def recur(baseVal: Int, combiner: (Int, Int)=>Int): Int=>Int = ???
```

It returns a traditional-style recursive function f: Int=>Int.

B. Use your recur combinator to implement a factoirial function.

5. Problem

Some programmers like to handle errors by throwing exceptions, others like to return the optional value None. Implement a combinator called deOptionize that takes a unary, option-returning function,

f, as input and converts it into a non-option returning function g that handles errors by throwing exceptions.

Use your combinator to convert the following function:

```
def parseDigits(digits: String): Option[Int] =
   if (digits.matches("[0-9]*")) Some(digits.toInt) else None
```