Sequence Control

To begin:

· In Eclipse create a Scala project called ControlLab.

· 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. You should use library functions whenever possible.

To end:

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

Problem 1

Elbonia has a simple sliding scale income tax scheme. Your tax rate is determined by your income according to the following table:

Income	Rate
< $20000	0%
< $30000	5%
< $40000	11%
< $60000	23%
< $100000	32%
>= $100000	50%

Implement a tax calculator for the Elbonians using Scala's match expression.

Your calculator should throw an InvalidIncome exception if the input is negative.

Problem 2

Write a procedure that draws an n by m rectangle:

```
drawRectangle(3, 4)
```

```
****
****
****
```

Problem 3

Write a procedure that prints the sums of all integers i and j for 0 < i < n and 0 < j < m:

```
printSums(4, 3)
1 + 1 = 2
1 + 2 = 3
2 + 1 = 3
2 + 2 = 4
3 + 1 = 4
3 + 2 = 5
```

Hint: Use a for loop with multiple generators and a guard condition.

Problem 4

Rewrite the following Java method in Scala. Try to preserve as much of the logic as possible.

```
class EscapeDemo {
     public void mystery() {
        for(int i = 0; i < 100; i++) {
           if (i % 3 == 0) continue; // skip if i divisible by 3
           if (i == 10) break; // terminate loop when i is 10
           System.out.println("i = " + i);
        }
        System.out.println("done");
     }
}
```

Problem 5

The following functions return optional doubles:

```
def root(x: Double): Option[Double] = if (x < 0) None else Some(math.sqrt(x))
```

```
def below10(x: Double): Option[Double] = if (x < 10) Some(x) else None
```

Use these functions to implement:

```
def pureRoot(x: Option[Double]): Option[Double] = root(y) if x = Some(y) else None
```

```
def pureBelow10(x: Option[Double]): Option[Double] = below10(y) if x = Some(y) else None
```

Use these to define

```
below10root(x: Option[Double]): Option[Double] = root(y) if x = Some(y) and y < 10 else
None
```

Problem 6

Assume the following declarations have been made:

```
var x = 10
```

```
lazy val y = math.log(-1)
```

Without using Scala, what are the values (with their types) of the following expressions, if the expressions contain errors, write "error":

```
if (false) 3
```

```
if (false) if (true) 3 else 4      // dangling else
```

```
if (x = 10) true else false
```

```
if (true) 1 else 1/0               // conditional eval
```

```
if (if (true) false else true) true else false
```

```
for(i <- 1 to 5) yield i * i
```

```
y + 10
```

```
println("100") // tricky!
true || 3 == 1/0                 // short circuit eval
false && math.log(-1) > 0        // short circuit eval
```

Now check your answers using the Scala interpreter.

Problem 7

Assume the following declarations have been made:

```
var x = 10
```

Without using Scala, what are the values (with their types) of the following expressions, if the expressions contain errors, write "error".

```
{1; 2; 3} + {4; 5; 6}
{1; 2; {3; 4; {5; 6} } }
if (false) {2; 3} else {4; {5; 6}} + {10; 20}
{10; 20} + if (false) {2; 3} else {4; {5; 6}}
{var x = 20; var y = x + 1; x + y}
{ var y = x + 1; var x = 5; var z = x + 1; x + y + z}
{var x = 3; var y = {var x = 9; x + 1}; x + y} // shadowning
{def tri(n: Int): Int = if (n == 0) 0 else n + tri(n - 1); tri(5) }
// recursive blocks
```

Now check your answers using the Scala interpreter. Ignore the warning messages, but not the error messages.