TUTORIAL 4

Question 2 in the exam paper focuses in on functions and their use in a functional approach. The second part typically involves exploring some of: Higher Order functions; partial application; closures; currying; folding and zipping.

1. Look at the following expressions:

```
def lambda = { (x: Int) => x + 1 }
val lambda = (x: Int) => x + 1
```

   Either of these will allow an expression such as *lambda(4)* to be evaluated:

   Describe what each expression does, and in what way they are different.

2. What will the values of *result1* and *result2* be after the following code runs? Describe the scope of the variable *incrementer* and the scope from which the function *closure* is called.

```
object MyObj {
    def main(args: Array[String]) {
        var incrementer = 1
        def closure = {x: Int =>  x + incrementer}
        val result1 = closure(10)
        incrementer = 2
        val result2 = closure(10)
    }
}
```

3. What will the value of *result* be after the following code runs? Describe the scope of the variable *incrementer* and the scope from which the function *closure* is called. Explain the feature of Scala that allows this code to work.

```
object MyObj {
    def main(args: Array[String]) {
        var incrementer = 1
        def closure = {x: Int =>  x + incrementer}
        val result = summation(10, closure)
    }

    def summation(x: Int, y: Int => Int) = y(x)
}
```

4. What will the type of *result* be after the following code runs? How can the function *add* be used to add the values 5 and 2 to give 7?

```
def add (x: Int) = {(y: Int) => x + y}
```

```
val result = add(5)
```

5. Describe the types of the parameters of the function *transform*. What is the value of *result* after the following code runs? *Note that the map method of List applies the specified function to each element of the list and returns another list. In the code it is called without parentheses, which is allowed in Scala, so the code inside the function transform is equivalent to list.map(f)*.

```
def transform (list: List[String], f: String => String) = {
  list map f
}

val result = transform (List("ABC", "XYZ", "123"), {x =>
     x.toLowerCase})
```

Write an expression using *transform,* with the same first parameter, that will return:

```
List(CBA, ZYX, 321)
```

6. Describe the types of *sumA*, *sumC* and *result* and the value of *result* after the following code runs

```
def sum(a: Int, b: Int, c: Int) = a + b + c
val sumC = sum(1, 10, _: Int)
val result = sumC(4)
```

7. Describe the types of *capital* and *result* and the value of *result* after the following code runs, using the *transform* function in question 5.

```
val capital = transform(_:List[String], {x => x.capitalize})
val result = capital(List("abc", "xyz", "123"))
```

8. Write a curried version of the function *transform* in question 5, and an expression that uses it to get the same result as in question 5.