String Processing

To begin:

- In Eclipse create a Scala project called StringLab.
- Add a worksheet to this project called stringSession
- 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. You should use operator symbols whenever possible.

Hint: it's helpful to know that Instances of the String class are implicitly converted into instances of:

scala.collections.immutable.StringOps

To end:

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

1. Problem

Write and test a simple palindrome detector:

```
isPal("rotator") => true
isPal("cat") => false
etc.
```

Note: isPal should ignore leading and trailing whitespace. It should also be case insensitive. It should work even if the string contains characters other than letters.

2. Problem

Enhance isPal by making a function isPal2 that ignore case, punctuation, and white space:

```
isPal2("A man, a plan, a canal, Panama!") => true
```

Hint: filter out undesirable characters, then use the function defined in #1.

3. Problem

Write and test a palindrome constructor:

```
mkPal("mars") => "marssram"
mkPal("3X@#") => "3X@##@X3"
```

4. Problem

Write a random word generator:

```
val a1 = mkWord()   => a1  : String = ltikizlrbmx
val a2 = mkWord()   => a2  : String = iceqyxdtcqx
val a3 = mkWord()   => a3  : String = dcjjqsecegi
val a4 = mkWord(20) => a4  : String = erlazfucnscevefzaaviv
```

Note: Words only contain lowercase letters. The average length should be approximately the average length of words in an English dictionary.

Hints:

Before your definition of mkWord type:

```
import scala.util.Random
```

Random is Scala's random number generator. It's a pre-defined object, though, not a class.

Given a random UNICODE, x, the expression x.toChar will convert it into the corresponding character.

For example:

```
scala> 0x61.toChar
res4: Char = a
```

Another hint: Scala functions can take default arguments:

```
def display(prompt: String = "=> ") {
   print(prompt)
}

scala> display()
=>
```

5. Problem

Write a random sentence generator:

```
val sen1 = mkSentence()  => sen1  : String = Onbdcevx ldqdhy xhikrfulbl m dxqfmmkrvy
hqvynsxfj gpkak rhpngvigp fqtsaxwiv dong.
val sen2 = mkSentence()  => sen2  : String = Usuljsyoo cavtwisgtx jokiodqm ln zpaeb cov
eezi foy | odepzev trdofvth wtnjhrwuet.
val sen3 = mkSentence()  => sen3  : String = Jbpnwlquh qm r jzeiwi bms z hoyy qogwgbdmw
lqxzuwnj y qwq fhib l.
val sen4 = mkSentence(5) => sen4  : String = Lsxa fvtqnlz jnyv xpowyjydoi ltqekjm.
```

Notes:

- Sentences begin with an uppercase letter and end with a period.
- mkSentence can take an argument, but otherwise uses a default argument.
- Using mkSentence how would you implement mkNovel? mkPoem? mkReply? mkPost?

6. Problem

Write a function that appends the first half of a string to its last half:

```
shuffle("abcdefghij") = fghijabcde
shuffle("abcdefghijk") = fghijkabcde
```

Hint: This would be a good time to learn about take and drop.

7. Problem

Write a function that counts the number of occurrences of a non-empty string in another:

```
countSubstrings("is", "Mississippi") => 2
```

Live dangerously, think about making countSubstring recursive!

8. Problem

Write a function that evaluates sums of two numbers:

```
eval("3.14+42")  = 45.14
eval("  -26  +  -49.99  ") = -75.99000000000001
add("21 * 43") = java.lang.Exception: missing operator
add("abc + 3") = NumberFormatException
```

Hints:

You can use take and drop to extract the numbers.

Here's how to convert strings to numbers:

```
scala> 4.toDouble
res15: Double = 4.0
```

Another trick is to use split:

```
scala> "23 + 42".split("\\s+")
res16: Array[String] = Array(23, +, 42)
```

The string "\\s+" is actually a regular expression pattern matched by all sequences of white space characters.

This allows you to deal with more complex expressions involving multiple additions. The downside is that it that split returns an array of strings. To access the elements of an array use the following syntax:

```
someArray(someIndex)
```

9. Problem

Modify eval so that it also computes products.

10. Problem

Write a function that returns the lexicographically first string in an array of strings:

```
first(Array("cat", "rat", "bat") = "bat"
```