Additional resource material:

# <https://www.tutorialspoint.com/scala/scala_if_else.htm>

<https://www.tutorialspoint.com/scala/scala_loop_types.htm>

<https://www.tutorialspoint.com/scala/scala_file_io.htm>

<https://www.tutorialspoint.com/scala/scala_exception_handling.htm>

**Use sets and maps**

// import scala.collection.immutable

object Hello

{

def main(args: Array[String]): Unit =

{

println("Hello Scala Programmer!")

// Odersky, page 47

// jetSet is a (default) immutable Set

var jetSet = Set("Boing", "Airbys")

jetSet += "Lear" // jetSet has been replaced

println(jetSet)

// Odersky, page 49

// movieSet is a mutable set

val movieSet = scala.collection.mutable.Set("Hitch", "Polter")

movieSet += "Shrek"

// movieSet has been modified

println(movieSet)

}

}

object findIt

{

def main(args:Array[String]); Unit =

{

// Odersky, page 50, 51

import scala.collection.mutable

val treasureMap = mutable.Map[Int, String]()

treasureMap += (1 -> "Go to island")

treasureMap += (2 -> "Find big X on ground")

treasureMap += (3 -> "Dig")

println(treasureMap(2))

// immutable is the default

val romanNumeral = Map(

1 -> "I", 2 -> "II", 3 -> "III", 4 -> "IV", 5 -> "V")

println(RomanNumeral(4))

}

}

**Exercises – Day02**

1. Write and test a function that accepts a string as an argument and prints out whether this string is a palindrome or not.

A **palindrome** is a string that reads the same forwards and backwards. In testing your string you should ignore blanks, punctuation and case differences (be case-insensitive). Be sure that you test multi-word strings such as (“Sit on a potato pan, Otis.”)

1. Write and test a function that accepts a numeric argument (a **limit**) and prints all the Fibonacci numbers less than or equal to this limit. The Fibonacci sequence is a sequence of numbers where the next number in the sequence is the sum of the previous two numbers in the sequence. The sequence looks like this: 1, 1, 2, 3, 5, 8, 13, … (it should not start with 0).
2. Write and test a function that accepts a string argument. This will be a long string containing multiple words. Print the string but with the words (not the characters) reversed. For example, the string “My name is jim” would be printed back as “jim is name my” (**not** as “mij si eman yM”)
3. Write and test a function that asks for the names of all the members in a club. However, we don't know how many members are actually in the club. Use a “while-loop” which will simply repeat until all the member's names have been entered. How will you communicate to the program that there are no more names to enter?
4. Write and test a function that accepts a string containing a password and checks for its validity. The rules for a valid password are:
5. At least 1 letter between [a-z] and 1 letter between [A-Z].
6. At least 1 number between [0-9].
7. At least 1 character from [$#@].
8. Minimum length 6 characters.
9. Maximum length 16 characters.
10. Write and test a function that asks the user to enter an integer greater than 1. The program will calculate and print the average of the list of numbers going from 1 up to and including the number the user entered.
11. Write and test a function that accepts a numeric argument , a **limit**, and finds all numbers between 1 and the limit where each digit of the number is an even number. The numbers obtained should be printed. Ex: 76 doesn’t pass the test (7 isn’t even) while 48 does pass and would be printed.
12. Write and test a function to get date, one day after a given date. You may ignore leap years and assume that February always has 28 days.  
     Example:

Input a year: 2016

Input a month [1-12]: 08

Input a day [1-31]: 23

The next date is [yyyy-mm-dd] 2016-8-24

1. The keys in a dictionary are guaranteed to be unique but the values are not. Create and test a function that accepts a single dictionary as an argument and returns the number of distinct values that it contains.
2. A *sparse vector* is a vector whose entries are almost all zero, such as Vector(1, 0, 0, 0, 0, 0, 3, 0, 0, 0). Storing all those zeros uses a lot of memory so a way to avoid that is to use a map to represent just the non-zero entries. For example, the vector shown earlier would be represented as Map(0 -> 1, 6 -> 3) because the vector that it represents has the value 1 at index 0 and the value 3 at index 6.

* The sum of two vectors is the element-wise sum of their elements. For example, the sum of (1, 2, 3) and (4, 5, 6) is (5, 7, 9). Write a function that that accepts two sparse vectors stored as maps and returns a new map representing their sum.
* The dot product of two vectors is the sum of the products of corresponding elements. For example, the dot product of (1, 2, 3) and (4, 5, 6) is 4 + 10 + 18 or 32. Write a function that calculates the dot product of two sparse vectors.