Assignment 1

Use the following sources to get familiar with Swift:

- https://developer.apple.com/documentation/swift
- https://www.swift.org
- https://www.hackingwithswift.com

Exercise 1 – Variables and Constants

Create two variables, *name* as a String, and *birthyear* as an Int, and a constant *country* as a String. Determine the *age* in years and whether birthyear was a leap year, and save it in another variable *isLeapYear*. Depending on this variable, print one of the following two messages:

- "My name is (name), I was born in (birthyear), which was a leap year, I am (age) years old, and I live in (country)."
- "My name is (name), I was born in (birthyear), which was not a leap year, I am (age) years old, and I live in (country)."

Read the values for name, birthyear, and country from the console and test the program with several different values. Please note that in Xcode you need to create a "Command Line Tool" for this exercise.

Exercise 2 – Temperature Converter

Write a Swift program that converts temperatures between *Celsius* and *Fahrenheit*. The program should ask the user to input five different temperature values (store them in an array) and the scale (Celsius or Fahrenheit), then perform the conversion and display the result for the five values. The formula for conversion is:

- Celsius to Fahrenheit: (C * 9/5) + 32
- Fahrenheit to Celsius: (F - 32) * 5/9

Exercise 3 – Student Grade Calculator

Write a program that calculates the average grade of a student based on five subject scores (entered by the user). Based on the average, the program should determine the student's grade as follows (use a switch-case statement for this):

- average ≥ 89: Grade 1
- average ≥ 76: Grade 2
- average ≥ 63: Grade 3

- average ≥ 50: Grade 4
- below 50: Grade 5

Exercise 4 – Rock, Paper, Scissors Game

Create a Swift program that allows the user to play the classic game of Rock, Paper, Scissors against the computer. The program should randomly select Rock, Paper, or Scissors for the computer (use an enum for this), and then compare it to the user's choice to determine the winner.

Exercise 5 – Structs and Classes

Implement a struct *Position* that represents a position in a graphical coordinate system and consists of two floating numbers *x* and *y*. Implement a class *Shape* that uses the struct for an instance variable *pos*. Implement an appropriate initializer as well.

Next, create two classes *Rectangle* and *Circle*, which inherit from Shape. In addition to the inherited position, a rectangle has a *width* and a *height,* and a circle has a *radius* (you can assume that every value is specified in m).

Now create 50 shapes (either a rectangle or a circle, based on a boolean random decision) and store them in an array called *shapes*. For rectangles use a position of x=0, y=0, a random width (between 100…200), and a random height (between 50…100). For circles use x=10, y=10, and a random radius between 5π and 10π.

Print the created objects in the shapes array to the console, to see what has been created. Implement a function *printShapes()* for this purpose, which iterates over the array and simply prints the object by implicitly calling the *description* property of the *CustomStringConvertible* protocol (equivalent to the *toString()* interface in Java). The description property returns appropriate strings of the corresponding object.

Exercise 6 – Objects

In this exercise we extend the solution of Exercise 5. First, we want to sort all objects in the array by the area (ascending). Use another property *area* for that purpose, which computes the area of the corresponding shape according to the appropriate area formulas for a rectangle and a circle. Print the sorted array to the console again by using the already implemented function *printShapes()*.

Moreover, we also want a nicer output of our program. Therefore, initialize a *NumberFormatter* (available in UIKit) in the Shape initializer and use it in the description and area properties of Rectangle and Circle (for width, height, radius, and area). We want to use

the German number format (i.e., comma as digit separator and dot as a digit grouping character) and two digits after the comma for all floating-point numbers.

Exercise 7 – Xcode

Use the Apple Developer Documentation[1] to answer the following questions regarding Xcode in detail (you should be able to demonstrate this in class):

a) What exactly is the *Info.plist* file?
b) How can we use images in an app and how to provide different images for devices with different screen resolution?
c) How can we configure the icon for an app?
d) How can we vary the color (for example) in respect to the device orientation?
e) How can we use different colors and images for the 'Dark Mode'?
f) How can we provide different texts for different localizations of the app?
g) What exactly is a *Build Target*?

Exercise 8 – UIKit Basics

Use the Apple Developer Documentation to answer/solve the following questions/tasks:

a) Create a new UIKit project ('Single View App') and explain the different files that are created for it.
b) Explain the following terms in detail:
    - What is a *View*, a *Control*, and a *ViewCotroller*?
    - What is a *Storyboard* and a Segue, and what can they be used for?
    - What is an *Action Method* and how to connect one in Xcode?
    - What is an *Outlet* and how to connect one in Xcode?
c) What is a *Layout Constraint*? Present and demonstrate a few examples for layout constraints (for size, spacing, and arrangement).
d) What is a *Navigation View Controller* and how can create one (for a meaningful example)?

[1] https://developer.apple.com/documentation/