4 Coding

4.1 Underlying Data / API

The data for the APIs are from "https://coronavirus-tracker-api.herokuapp.com/v2/locations" and "https://www.travel-advisory.info/api". The implementation of the current CoVid numbers turned out to be much easier, as by dynamically changing the country ID at the end of the link, different countries and their Corona could be accessed.

```
let url = "https://coronavirus-tracker-api.herokuapp.com/v2/locations?country_code=\(countryID)"
```

This made it possible to use an API function and the same API struct for all countries:

```
// MARK: API Func Corona Numbers
 func getData(from url: String) {

    let task = URLSession.shared.dataTask(with: URL(string: url)!, completionHandler: { data, response,
        error in

        guard let data = data, error == nil else {
            print ("something went wrong")
            return
        }
        var latest: Response?
        do {
            latest = try JSONDecoder().decode(Response.self, from: data)
        }
        catch {
            print ("failed to convert \(error.localizedDescription)")
        }

        guard let json = latest else {
            return
        }
        print ("Confirmed:\(json.latest.confirmed)")
        print("Recovered: \(json.latest.recovered)")
        print("Deaths: \(json.latest.deaths)")

        LabelDisplay.confirmed = Int(json.latest.confirmed)
        LabelDisplay.recovered = Int(json.latest.recovered)
        LabelDisplay.deaths = Int(json.latest.deaths)
    })
    task.resume()
}
```

The code for the API query function was implemented with the support of a YouTube tutorial (https://www.youtube.com/watch?v=sqo844saoC4).

```swift
//MARK: CORONA NUMBERS STRUCT

struct Response: Codable {
    let latest: MyResult
}

struct MyResult: Codable {
    let confirmed: Int
    let deaths: Int
    let recovered: Int
}

struct LabelDisplay {
    static var confirmed: Int = 0
    static var recovered: Int = 0
    static var deaths: Int = 0
}
```

The second API was a little more time-consuming, as the country ID could not be dynamically adjusted. Country-specific URLs, functions and structures had to be created in order to be able to access the data.

```swift
let url5 = "https://www.travel-advisory.info/api?countrycode=IT"
let url6 = "https://www.travel-advisory.info/api?countrycode=NO"
let url7 = "https://www.travel-advisory.info/api?countrycode=GR"
let url8 = "https://www.travel-advisory.info/api?countrycode=DK"

getDataSecondIT(from: url5)
getDataSecondNO(from: url6)
getDataSecondGR(from: url7)
getDataSecondDK(from: url8)
```

```swift
//MARK: ITALY
private func getDataSecondIT(from url5: String) {

    let task = URLSession.shared.dataTask(with: URL(string: url5)!, completionHandler: { data, response,
        error in

        guard let data = data, error == nil else {
            print ("something went wrong")
            return
        }
        // data
        var advisory: ResponseIT?
        do {
            advisory = try JSONDecoder().decode(ResponseIT.self, from: data)
        }
        catch {
            print ("failed to convert \(error.localizedDescription)")
        }

        guard let json = advisory else {
            return
        }


        print (json.data.IT.advisory.message)
        AdvisoryMessageIT.message = json.data.IT.advisory.message

    })

    task.resume()
}
```

```
//--------------IT-------
struct ResponseIT: Codable {
    let data: DataIT
}

struct DataIT: Codable {
    var IT: CountryCodeIT
}
struct CountryCodeIT: Codable {
    let advisory: AdvisoryIT
    }

struct AdvisoryIT: Codable {
    let message: String
    }
//---------GR-------
struct ResponseGR: Codable {
    let data: DataGR
}

struct DataGR: Codable {
    var GR: CountryCodeGR
```

The current CoVid19 numbers and instructions have been implemented by a total of four countries (Greece, Italy, Denmark and Norway).

4.2 Used Techniques

The questions should all be loaded onto a ViewController, which meant that a question struct array had to be created, which changes the displayed question label after pressing the button.
So that the result depends on the questions answered, a score was added that increases by +2 for the answer yes and by +1 for no. This means that a country can be assigned to a certain score range. With the help of the score, a progress bar could also be inserted into the ViewController, which increased after each question. The end score was then loaded to the ResultViewController using a segue and served as the overall output of the answered questions. This made it very easy to display the correct parameters of the labels with a switch function. The MapViewKit was used to display the map. The exact view was set up with two let constants and could easily be changed by the variables latitudeCountry and longitudeCountry from the outputLandName switch function.

```
//MARK: set mapview location
let location = CLLocationCoordinate2D(latitude: latitudeCountry, longitude: longitudeCountry)
let region = MKCoordinateRegion(center: location, span: MKCoordinateSpan(latitudeDelta: 12,
    longitudeDelta: 12))

self.mapView.setRegion(region, animated: true)
```

Since the ResultViewController needed some time to display the results immediately, a LoadingScreen was implemented in an extra Swift file and called from the transition to the ResultViewController. The code was taken from a forum post and integrated into the app:
https://stackoverflow.com/questions/27960556/loading-an-overlay-when-running-long-tasks-in-ios. The LoadingScreen was especially necessary because of the API functions, as it took some time to obtain information. The assignment of the Corona Numbers and Advisory Message to the result country was therefore outsourced to a second switch function and called separately.

4.3 Functions and Method description

The questions are created using the createQuestionObjects function, which saves entries in a separate question class:

```
//MARK: - Create question objects

func createQuestionObjects() {
    let question1 = Question(text: "Do you prefer a warm climate?")
    question1.answer1 = "Yes"
    question1.answer2 = "No"
    //question1.answer3 = "Antwort3"
    question1.correctAnswerTag = 1

    let question2 = Question(text: "Should it be an inexpensive vacation?")
    question2.answer1 = "Yes"
    question2.answer2 = "No"
    //question2.answer3 = "Antwort3"
    question2.correctAnswerTag = 1
```

After pressing the answer button, the functions checkAnswer() and nextQuestion() are called. The checkAnswer() function compares the answer with the correct announcement day and increases the score either by +2 for yes and by +1 for no, and the questionNumber, which calls up the next question, is increased by +1. The nextQuestion() function ensures that the questions and answers are displayed correctly and counts. After the last question, it opens the ResultViewController and passes on the entire score using a segue.

```
@IBAction func answerButton_Tapped(_ sender: UIButton) {
    let answerTag = sender.tag

    checkAnswer(answerTag: answerTag)

    nextQuestion()
}

func checkAnswer(answerTag: Int) {
    if answerTag == questions[questionNumber].correctAnswerTag {
        print ("Positiv")
        score += 2
    } else {
        print ("Negativ")
        score += 1
    }
    questionNumber += 1
}

func nextQuestion() {
    if questionNumber < questions.count {
        questionLabel.text = questions[questionNumber].questionText
        answerButton1.setTitle(questions[questionNumber].answer1, for: .normal)
        answerButton2.setTitle(questions[questionNumber].answer2, for: .normal)
        //answerButton3.setTitle(questions[questionNumber].answer3, for: .normal)

        updateUI()
    } else {
        self.performSegue(withIdentifier: "resultGame", sender: self.score)
        self.view.showBlurLoader()
    }
}
```

The progress bar is displayed correctly using the updateUI function and changes its length depending on the number of questions answered.

```
// MARK: display score
func updateUI() {
    scoreLabel.text = "Score: \(score)"
    questionCountLabel.text = "\(questionNumber + 1) / \(questions.count)"
    let widthIphone = self.view.frame.size.width
    let widthProgressCountView = Int(widthIphone) / questions.count

    progressBarView.frame.size.width = CGFloat(widthProgressCountView * (questionNumber + 1))

}
```

By overwriting the prepare function, the final score and the individual API Advisory Messages could be passed on to the ResultViewController.

```
//MARK: score/api segue to resultViewcontroller
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    var vc = segue.destination as! resultViewController  Variable 'vc'
    vc.questionscore = score
    vc.messagePassedIT = AdvisoryMessageIT.message
    vc.messagePassedNO = AdvisoryMessageNO.message
    vc.messagePassedGR = AdvisoryMessageGR.message
    vc.messagePassedDK = AdvisoryMessageDK.message
}
```

The total question score served as the basis for displaying the correct result. With the help of the switch functions outputLandName() and outPutMessage(), the result can be displayed using the question score. These functions determine the ResultCountryLabel, the MapView, the corona number and advisory message API.

```
func outputLandName() {
    switch questionscore {
    case 15...18:
        nameCountry = "Denmark"
        latitudeCountry = 56.26
        longitudeCountry = 9.50
        self.view.showBlurLoader()
        DispatchQueue.main.asyncAfter(deadline: .now() + 1.0) {
            self.getData(from: CountryFunctionDK.url)
            self.view.removeBluerLoader() }
    case 20...24:
        nameCountry = "Norway"
        latitudeCountry = 64.57
        longitudeCountry = 17.88
        self.view.showBlurLoader()
        DispatchQueue.main.asyncAfter(deadline: .now() + 1.0) {
            self.getData(from: CountryFunctionNO.url)
            self.view.removeBluerLoader()               }
    case 26...29:
        nameCountry = "Italy"
        latitudeCountry = 41.29
        longitudeCountry = 12.57
        self.view.showBlurLoader()
        DispatchQueue.main.asyncAfter(deadline: .now() + 1.0) {
            self.getData(from: CountryFunctionIT.url)
            self.view.removeBluerLoader() }
    default:
        nameCountry = "Greece"
        latitudeCountry = 38.27
        longitudeCountry = 23.81
        self.view.showBlurLoader()
        DispatchQueue.main.asyncAfter(deadline: .now() + 1.0) {
            self.getData(from: CountryFunctionGR.url)
```

```
//MARK: Func outputLandName
func outPutMessage() {
    switch questionscore {
    case 15...18:
        print ("case1")
        AdvisoryMessageLabel.text = messagePassedDK
        countryID = "DK"
    case 20...24:
        print ("case2")
        AdvisoryMessageLabel.text = messagePassedNO
        countryID = "NO"
    case 26...29:
        print ("case3")
        AdvisoryMessageLabel.text = messagePassedIT
        countryID = "IT"
    default:
        print ("case4")
        AdvisoryMessageLabel.text = messagePassedGR
        countryID = "GR"
    }
}
```