1) **Xcode & Swift: Introduction**

1) **Variables and Constants**

1) Is it possible to modify a mutable array that is an element of an immutable array? 
   No, because arrays are value types themselves and become immutable when added to a constant value type.

2) Is it possible to modify a custom class that is an element of an immutable array? 
   Yes, variable properties of the custom class can be mutated.

3) The following code snippet shows the definition of a class and a structure. Variables declared inside classes and structures are called properties, which can also be declared variable or constant. In the assignments in the code-snippet, which properties can be modified after initializing and assigning the respective objects?
   ```swift
   class Person {
       var age: Int
       let name: String
   }

   struct Car {
       var installedTires: String
       let engineSize: Int
   }
   
   let mouseImmutable = Person(age: 17, name: "Mouse")
   var mouseMutable = Person(age: 17, name: "Mouse")
   
   let fantasyCar1 = Car(installedTires: "Winter", engineSize: 8700)
   var fantasyCar2 = Car(installedTires: "Winter", engineSize: 8700)
   
   mouseImmutable.age
   mouseMutable.age
   fantasyCar2.installedTires
   ```

2) **Swift Types**

1) List all possibilities to declare an empty dictionary. What are the difference between them?
   - `[String: String]()` → already initialized
   - `Dictionary<String, String>()` → already initialized
   - `[String: String]` → type declaration
   - `Dictionary<String, String>` → type declaration

2) What ways exist to add a key-value pair to a given mutable dictionary?
   a) `mutableDict["key"] = "value"
   b) `mutableDict.updateValue("anotherValue", forKey: "anotherKey")`

3) A set contains several values is given. Print all elements of the set with their respective index number of retrieval.
   ```swift
   var mutableSet: Set<String> = 
   let (index, value) in mutableSet.enumerate() {
       print("\(index): \(value)")
   }
   ```

3) **Functions, Methods and Closures**

   a) Find a value in an array without using for or while loops.
   ```swift
   let array = 
   let result = array.filter { (element) -> Bool in
       return element == "Third"
   }
   ```
1) Declare a sort function that takes an array of integers as input, a closure to specify the comparison algorithm and that returns an ordered array.

```swift
func own_sort(inputArray: Array<Int>, isOrderedBefore: (firstElement: Int, secondElement: Int) -> Bool) -> Array<Int> {
    var sortArray = inputArray
    for i in 1...(sortArray.count-1) {
        for j in 1...(sortArray.count-i) {
            if isOrderedBefore(firstElement: sortArray[j-1], secondElement: sortArray[j]) {
                (sortArray[j-1], sortArray[j]) = (sortArray[j], sortArray[j-1])
            }
        }
    }
    return sortArray
}
```

```swift
let unsortedArray = [3, 7, 4, 20, 56, 17, 0, 5, 9, 100, 1]
let sortedArray = own_sort(unsortedArray) { (firstElement, secondElement) -> Bool in
    return firstElement > secondElement
}
```

2) Implement a modification of the previous function that directly operates on the input array and has no return value.

```swift
func own_inout_sort(inout inputArray: Array<Int>, isOrderedBefore: (firstElement: Int, secondElement: Int) -> Bool) {
    for i in 1...(inputArray.count-1) {
        for j in 1...(inputArray.count-i) {
            if isOrderedBefore(firstElement: inputArray[j-1], secondElement: inputArray[j]) {
                (inputArray[j-1], inputArray[j]) = (inputArray[j], inputArray[j-1])
            }
        }
    }
}
```

```swift
var unsortedInoutArray = unsortedArray
own_inout_sort(&unsortedInoutArray) { (firstElement, secondElement) -> Bool in
    return firstElement < secondElement
}
```

4) Classes, Structures and Enumerations

1) Implement the oppositeDirection function of the enumeration example on lecture slide 23.

```swift
func oppositeDirection() -> Directions {
    switch self {
    case .North: return South
    case .South: return North
    case .West: return East
    case .East: return West
    }
}
```

2) Add a depth property to the ExampleStructure on lecture slide 28 and add a new method that returns the volume.

```swift
struct ExampleStructure {
    var width: Int
    var height: Int
    var depth: Int
    func plane() -> Int {
        return width * height
    }
}
```
func volume() -> Int {
    return width * height * depth
}

3) Add immutable properties to the ExampleStructure that represent the structure's origin in a 3D coordinate system.

struct ExampleStructure {
    var width: Int
    var height: Int
    let x: Int
    let y: Int
    func plane() -> Int {
        return width * height
    }
}