**Exercises**

**Built-In Collections**
Open CollectionMethods.playground. You will find sample dictionary where keys are names of students and values are arrays of remarks for them:

```
let dict = [ "Samanta Malinowska"    : [ 1, 3, 2, 4, 4, 3, 3, 2 ],
             "Brian Nowak"           : [ 5, 5, 4, 3, 2, 1, 5, 3 ],
             "Jessica Kowalska"      : [ 5, 3, 3, 4, 5, 6, 4, 4 ],
             "Justin Lewandowski"    : [ 1, 2, 3, 3, 5, 1, 2, 4 ] ]
```

Perform following exercises using built-in methods from `Sequence` / `Collection` protocol (i.e. `map`, `filter`, `reduce`, etc.).
NOTE: Every single exercise should be solved with 1 line of code (even if it may not be highly readable).

1. Create an array of tuples where first element of tuple is the student name and second is his max remark.
   (*) In Swift 4 and Xcode 9, create dictionary where keys are student names and values are their max remarks.
2. Calculate sum of all remarks.
3. Calculate mean of all remarks
4. Return list of names of students that does not have any 1 - with the exception for first remark. In other words, having 1 as a first remark is acceptable if the student does not have any more 1's.
   (*) Sort the list by student's surnames. You can assume that every student's name has following form: `"<firstName><space><surname>"`

**Sequence & Collection**
1. Create **generic** `StackProtocol protocol` with two methods: `push` and `pop`.
2. Create implementation of this protocol -> generic `Stack<T>` struct. You can use array as an internal storage for the stack.
   (*) Additionally make the type conform to two more protocols:
   `a. ExpressibleByArrayLiteral`
   `b. CustomDebugStringConvertible`
3. Make `Stack` struct conform to `Sequence` protocol. Make sure `Stack` is NOT an unstable sequence.
4. Create `FibonacciSequence` (1, 1, 2, 3, 5, 8, 13, ...), make it conform to `Sequence`.
   (*) You can try to achieve this by not creating dedicated `FibonacciIterator` type, but by using `AnyIterator`.
5. If you implemented it correctly, `FibonacciSequence` is infinite. What happens if you call `reversed()` method on it?
6. Make `Stack` struct from points 1 - 3 conform to `Collection`.

**Specialized Collections**
1. Implement binary search algorithm so that it is available for as many collections as possible. It should be the method in `Collection` that returns optional index of the element (or `nil` if not found).
   ASSUMPTIONS / REQUIREMENTS:
   Assume the collection is sorted by the time of calling the method.
   The method should have O(log n) complexity where n is the `count` of the `Collection`.
   HINTS:
   a. Do you need any specialized collection (take performance requirements into account)?
   b. Do you need any requirements imposed on the elements / indices of the collection?

**Lazy Collections**

1. Reuse the dictionary from Built-in Collections group (i.e. the one with students and their remarks). Inspect what is the **type** (you can the type by Alt-click on the variable) of:
   a. `keys` property
   b. `values` property
   c. result of performing `dictionary.keys.map` (e.g. creating array of counts of characters in students' names)
   d. result of performing `dictionary.values.filter` (e.g. filtering out remarks that contain at least one 1)
2. print results from point 1.c and 1.d.
   a. Do you get readable results?
   b. Access first element from each result
   c. Turn these results into arrays, using dedicated initializer from `Array`.

**Ranges**

1. Create sample:
   `a. Range`
   `b. ClosedRange`
   `c. CountableRange`
   `d. CountableClosedRange`
2. Fill the following table with name of Range type OR with true / false

| **Expression** | `..<5.0` | `...5.0` | `5.0...` |
|---|---|---|---|
| **Type of range** | | | |
| **Contains 4.0 ?** | | | |
| **Contains 5.0 ?** | | | |
| **Contains 6.0 ?** | | | |

3. Fill the following table with true/false by guesses. Then, verify your guesses (either by checking documentation or directly in code).

| Type | **`Seuqence`** | Finite **`Sequence`** | **`Collection`** |
|---|---|---|---|
| **`Range`** | | | |
| **`ClosedRange`** | | | |
| **`CountableRange`** | | | |
| **`CountableClosedRange`** | | | |
| **`PartialRangeUpTo`** | | | |
| **`PartialRangeThrough`** | | | |
| **`PartialRangeFrom`** | | | |
| **`CountablePartialRangeFrom`** | | | |

**Slices & SubSequences**

1. Create some dummy class that has one property to identify itself (preferably of `String` type) and prints a message (including the identifier) when being `deinit`ialized.
2. Create var optional array of the dummy class and provide some initial data for it (at least 3 objects)

3. Get slice of the array e.g. `dummyArray?[1...2]` and it assign to some variable (name it `slice`). What is the type of this variable?
4. What will happen if you do `slice?[0]` ?
5. Nilify the array variable (NOT `slice`). Which objects get deinitialized?
6. Replace some object in the `slice`. Which objects get deinitialized?
7. Create sample dictionary and get some slice of it. What is the type of slice of `Dictionary`?
8. In Slices.playground you will find a function that checks if all elements of array are equal:

   ```
   func allElementsEqual(array: Array<Int>) -> Bool {
       guard let firstElement = array.first else {
           // Empty array is considered as having all elements equal
           return true
       }
       for element in array.dropFirst() {
           if element != firstElement {
               return false
           }
       }
       return true
   }
   ```

   You should not modify function body (i.e. implementation) while doing a) - c). NOTE: You can impose some restrictions on generic types used by `Collection` or `Sequence`.
   a. Improve the function so that it could be called on types like `Double` and `String`
   b. Improve the function from point a) so that it can also accept slices
   c. Change the function to not be global, but rather defined in extension on `Collection`.
   d. (*) Modify function body so that you can define the function in extension on `Sequence`.
9. Print, what is the type of:
   ```
   a. Array<String>.SubSequence
   b. Set<Int>.SubSequence
   c. Dictionary<Double, Bool>.SubSequence
   ```

**Hashable Requirement**
1. Implement scenario of breaking dictionary. In order to do it:
   - Create some class with some var property/-ies.
   - Make the class conforming to `Hashable`. Calculate `hashValue`, basing on some of the properties.
   - Create sample instances of the class.
   - Put these instances as keys of some dictionary.
   - Modify one of the property that `hashValue` depends on in one of the sample instances.
   - Check if the instance is still reported as being contained within dictionary.

**Optionals**
1. Create sample array of sth (`Int`s, `String`s, whatever you want) that also contains `nil`s. What is the type of the array? What is the type of `.first` property?
2. Create sample dictionary where some of the values are `nil` (keys could be anything). What is the type of the dictionary? What is the type of `dict[<key>]`?
3. Is it possible to create dictionary where one (but no more) of the **keys** is `nil`? Find out by trying to create such dictionary.
4. (*) Let's assume we want to implement `reduce1` function in extension on `Array`. The function works just like `reduce`, but instead of taking an initial value, it uses the first element in the array. If the array is empty, `reduce1` returns `nil`. Implement `reduce1` function, using mapping of optionals. HINT: You can try to implement this function without using optional mapping first, and then modifying the function to use optional mapping. Example calls:

```
["a", "b", "c"].reduce1(+)  // Optional("abc")
([] as [String]).reduce1(+)    // nil
```

5. (*) Implement function that checks whether two arrays of optional items are equal. Type wrapped by optional should be the same in both arrays. Example calls:

   ```
   optionalArraysEqual(["a", "b", nil], ["a", "b", nil]) // true
   optionalArraysEqual([nil] as [Int?], [nil, nil] as [Int?]) // false
   ```

**Copy-on-write**

1. Create variable array of `String`s and add `didSet` property observer (yes, you can define property observers not only on properties, but on normal variables as well) that `print`s something, when the variable is changed. Create similar variable array of `NSMutableString`s (with `didSet` as well). Call `<array>[0].append("someString")` on both arrays. Which didSet fired? NOTE: The results of this exercise will be different in Swift 3 and Swift 4.
2. Use some pure Swift class alongside with `isKnownUniquelyReferenced()` function to create `StackCOW` struct that conforms to `StackProtocol` from Sequence & Collection group and has copy-on-write semantics.
   HINT: You can use following code as a test case:

   ```
   var stack = StackCOW<Int>()
   stack.push(1)
   stack.push(2)

   var copy = stack // References should be shared
                    // - no actual copy should be done yet

   copy.push(3) // Copy on write should fire
                // - actual copy should be performed here

   stack.pop() // Returns 2
   copy.pop() // Returns 3
   ```

   NOTE: Of course using simple struct without COW will pass the test case, but that is not desired in this exercise :)

**Generics**

1. Implement two versions of `isSubset(of otherSequence: XXX)` method on `Sequence`. Each method should be defined in separate `extension`.
   There should be two specialized versions of the method for:
   a. first version should  take array of elements that implement `Hashable`
   b. second version should take array of elements that does not implement `Hashable`, only `Equatable`
2. Check if the correct version of your method is picked, by calling `isSubset(of:)` twice: one with array of objects that implement `Hashable` and second with array of objects that implement `Equatable`
   HINT: Create your own dummy type that is `Equatable`, but not `Hashable` OR use `UIViewAnimationOptions`.
3. Open Generics.playground. You will find an `extension` on `Collection` that bubble sorts the collection. However, the function does not compile. Without modifying implementation (i.e. function body) - but only by adding generic constraints to extension, make it compile.
   a. (*) After you make it compile, add some more constraints so that the bubble sort is available only for collections that guarantee O(n^2) execution time. HINT: In LinkedList.playground you will find a `MutableCollection` that will NOT execute `bubbleSort` in O(n^2) time (you can copy the code and find out ;) )

4. Check if you can extend `Array` so that it conforms to `Equatable` if type of `Element`s it holds conforms to `Equatable`.
5. Try to assign system `swap` function to some variable. Is it possible?
    a. Try to create `intSwap` variable that holds a function that is able to swap two `Int`s. Can you assign `swap` to `intSwap` variable now?

**Protocols**

1. Implement sample scenario for static VS dynamic dispatch problem:
    a. Create `Bird` (or `WingsMovable` if you are a Swift purist :) ) protocol with 1 method: `moveWings()`.
    b. Add and extension to `Bird` protocol that defines: default implementation of `moveWings()` (it could simply `print` something) as well as `fly()` method (let's say it calls `moveWings()` method 10 times).
    c. Create `Penguin` class that conforms to `Bird` and provides custom implementation of both `moveWings()` and `fly()` methods (both can simply `print` some custom messages).
    d. Create sample `Penguin` instance and assign it to some variable. Call `fly()` on the variable. Which method get called - default or custom?
    e. Provide explicit type for the variable from point d. -> set it type to be `Bird` (instead of default, implicit `Penguin`). Can your `Penguin` fly now? :)
2. Implement type eraser for `StackProtocol` protocol (from Sequence and Collection group) -> `AnyStack`.
3. What is size of `Int` variable? Use `MemoryLayout.size(of:)` method to find out. Does the size change if you assign the variable to other variable that has the type of some protocol (like `CustomStringConvertible`; unfortunately `SignedInteger` (Swift 3) / `BinaryInteger` (Swift 4) can not be used as type, because it has `Self` requirement).
    a. Perform similar check for `String`.

**Mirrors**

1. Open "Mirrors.playground". You will find a bunch of structures defined and some sample instances created. Using Mirror API, Implement `findPropertyWithName(_:in:)` so that the last line of the playground returns 2020.
2. (*) Implement sample swizzling in Swift on some classes.
3. (**) Implement dynamic adding of method to some Swift class.