CS 11 Swift/iOS

Lab 1: Swift tour

CS 11 course materials for students

View the Project on GitHub

Some Guidelines:

It is often helpful to prototype and experiment in Playgrounds, but when you submit your work, it should be in a Swift file, Lab1.swift, which should compile using `swift Lab1.swift` (which should have no output - for now, Playgrounds will be the easiest way to test your code).

Part A - Numeric types

For each of these, use a Swift playground to evaluate each expression, and write the result in a comment, including the type.

Example: 1 + 1 = 2 (Int)

1. 1.0 + 1
2. 2 * 24
3. 2.0 * 3
4. 4 / 2
5. 4 / 3
6. 4 / 3.0

Part B - Basic functions

Write all of the following functions.

I've specified method signatures for the functions in this lab, make sure you follow these guidelines so your code can be automatically tested. You'll notice that there are some functions with the same name, but different signatures - this is possible in Swift and standard throughout UIKit and other Swift frameworks - I don't find it to be a great practice but it can be convenient and is good to know about.

1. Write a function called `getGreeting` which takes one string argument and returns a greeting string. This function should take a `name` String and return the String "Hello, , I am your robot friend!"

This function should be able to be called as follows:

```
getGreeting(name: "Jake Gyllenhal")
```

1. Write a function called `thermometer` which takes one integer argument, and returns a string describing the weather. Should take a number of degrees, and return the string "It is degrees outside."

This function should be able to be called with no argument labels, like this:

```
thermometer(77)
```

1. Write a function called `double` which takes one *optional* integer argument, and either returns nil, or two times the integer passed in, depending on whether or not the argument is `Some` or `None`.

This function should be able to be called with no argument labels, like this:

```
double(77)
```

1. Write a function called `sum` which takes one Int array argument, and returns the sum of all the integers in the array.

This function should be able to be called with no argument labels, like this:

```
sum([1, 2, 3, 4, 5])
```

1. Write a function called `sum` which takes an array of optional Ints, and returns the sum of all the integers in the array.

This function should be able to be called with no argument labels, like this:

```
sum([1, nil, 2, 3, nil, 4, 5, nil])
```

1. Write a function called `sumOfOdds` which takes an array of Ints, and returns the sum of all the odd integers in that array.

This function should be able to be called with no argument labels, like this:

```
sumOfOdds([1, 2, 3, 4, 5])
```

1. Write a function called `average` which takes any number of Double arguments, and returns the average of them.

This function should be able to be called with no argument labels, like this:

```
average(1.0, 5.5, 3.4)
```

1. Write a function called `minimum` which takes an array of Ints, and returns an optional Int which is nil if the array argument is empty, or the smallest Integer in the array if the array is non-empty.

This function should be able to be called with no argument labels, like this:

```
minimum([4, 2, -1])
```

1. Write a function called `minimum` which takes an array of Ints and a separate Int, with the argument label `atMost`, which returns the minimum Int in the array, or `atMost`, whichever is smaller.

This function should be able to be called with only an argument label for

`atMost`, like this:

```
minimum([4, 2, 5], atMost: 1)
```

1. Write a function called `firstIndex`, which takes an Int argument labelled `of` and an Int array labelled `from`, and returns either the first index of `of` in the array `from`, or nil if `from` does not contain `of`.

This function should be called with both argument labels, like this:

```
firstIndex(of: 4, from: [1, 6, 4, 3])
```

Part C - Classes and Methods

For this part, you will write a simple class called `TaskList`. It should store a dictionary mapping tasks (strings) to priorities (integers). It should support these methods:

- `add(task: priority:)` - adds a task with the given priority to the class.
- `complete(task:)` - mark the given task as complete (remove it from the list)
- `tasks()` -> Returns an alphabetical array of the descriptions of each task in the list.
- `mostUrgent()` - returns a String optional which is the description of the task with the highest priority value.

You shouldn't need to write an initializer for this class, but if you do, make sure it can be called with no arguments for an empty to-do list. As you can see, this class is very simple and limited in usefulness - you can't change the priority of a task, or have subtasks or anything complicated. We will continue to add more features and use this class in future labs.