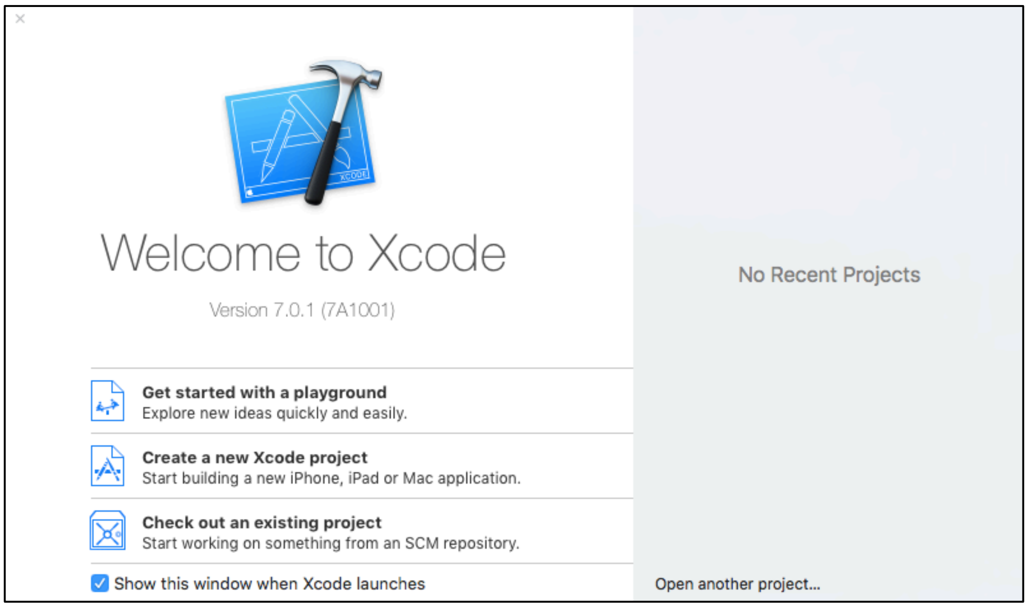
Practical 01: Using Playground

In this lab, we will walk you through the basics of creating a new Playground project and learning more about Swift without having to write a full-blown iOS app. This Playground practical is designed to allow you to settle into the Swift language by implementing some of these codes.

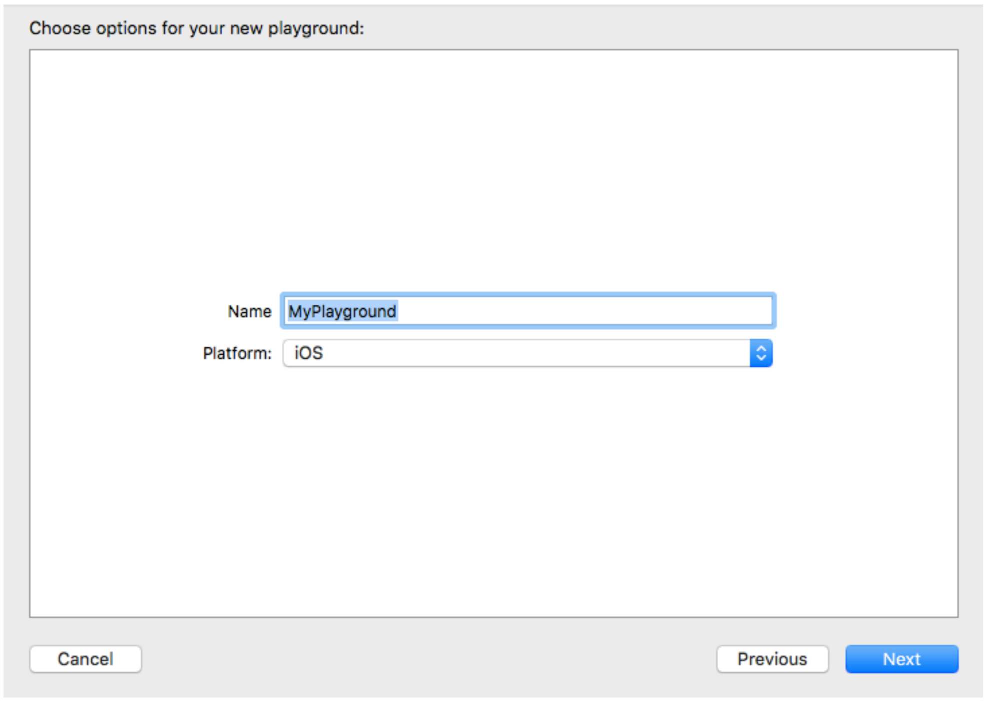
Note: This requires XCode 8 and Swift 3.0.

Section 1: Working with Playground

1. When you first launch Xcode, you are presented with a welcome screen as shown below.


Welcome to Xcode
Version 7.0.1 (7A1001)
Get started with a playground
Explore new ideas quickly and easily.
Create a new Xcode project
Start building a new iPhone, iPad or Mac application.
Check out an existing project
Start working on something from an SCM repository.
Show this window when Xcode launches
No Recent Projects
Open another project...

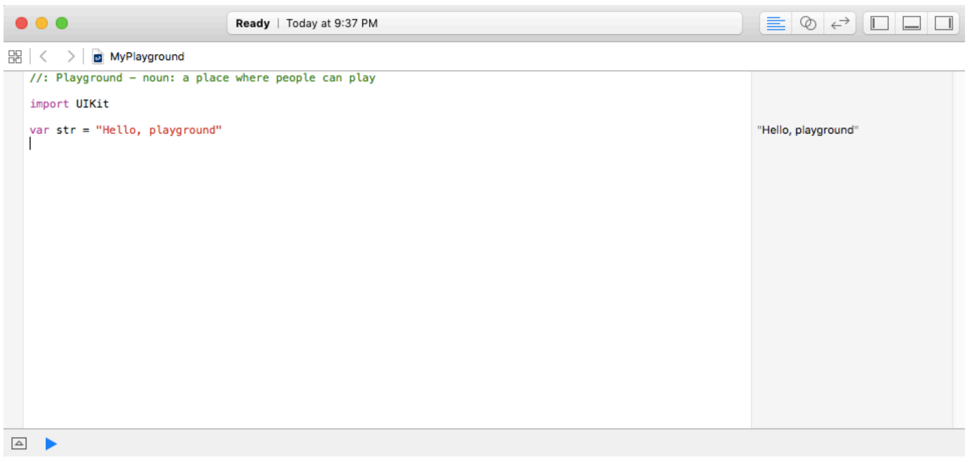
2. To write code using playground, click **Get started with a playground**.
3. In the next screen, click on the **Next** button.


Choose options for your new playground:

Name: MyPlayground

Platform: iOS

4. Select the folder to save your playground, and click the **Create** button.
5. Xcode now displays your playground.



```
//: Playground - noun: a place where people can play

import UIKit

var str = "Hello, playground"
```

```
"Hello, playground"
```

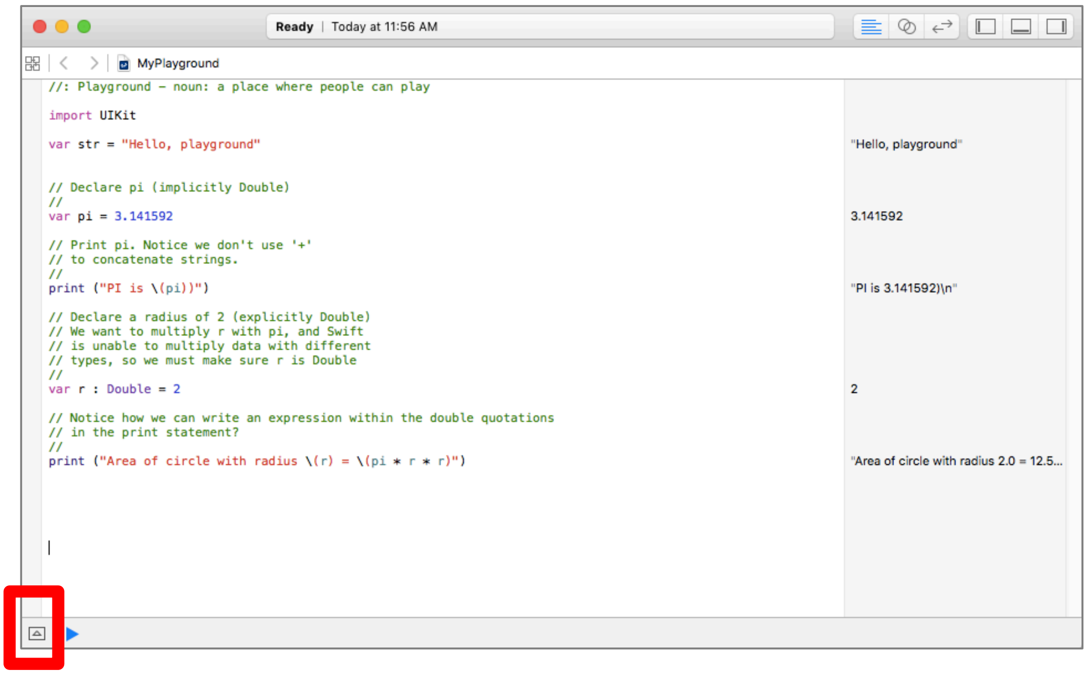
Variables and Data Types

1. We will create some variables of different types and print them to the screen. Append the following code to the end of your Playground code.

```
import Foundation
```

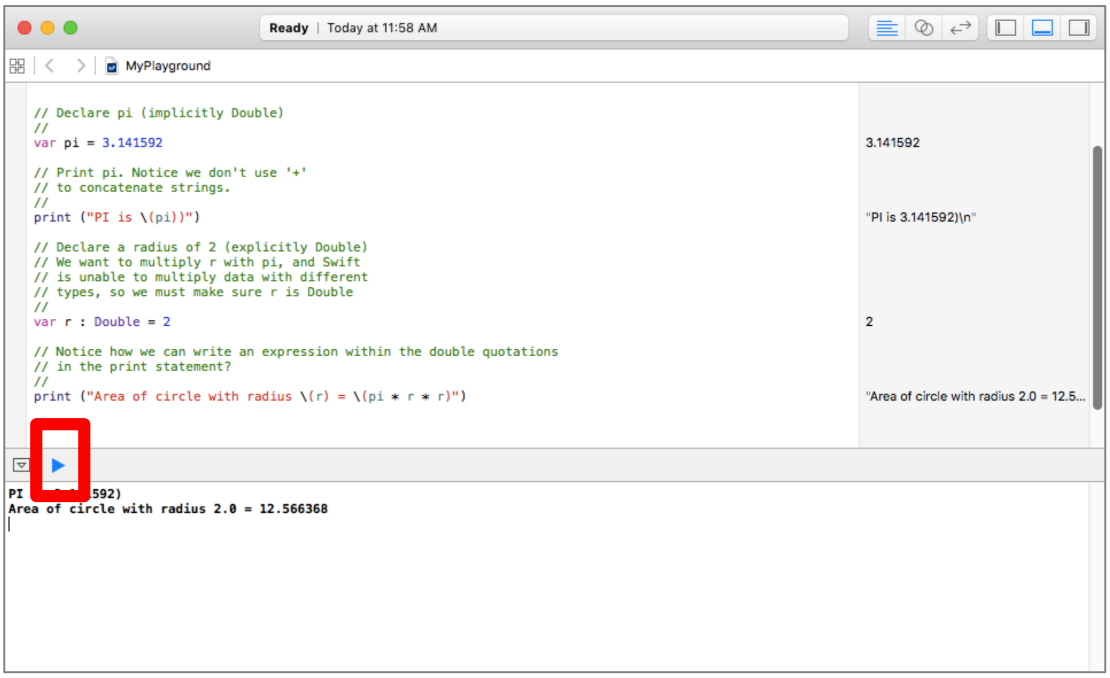
```
// Declare pi (implicitly Double)
//
var pi = 3.141592

// Print pi. Notice we don't use '+'
// to concatenate strings.
//
print ("PI is \(pi)")

// Declare a radius of 2 (explicitly Double)
// We want to multiply r with pi, and Swift
// is unable to multiply data with different
// types, so we must make sure r is Double
//
var r : Double = 2

// Notice how we can write an expression within the double quotations
// in the print statement?
//
print ("Area of circle with radius \(r) = \(pi * r * r)")
```

2. Before we run the code, we want to make sure we can see the output generated by your code. To do so, click on the highlighted icon at the bottom left corner in your Playground window.



```
//: Playground - noun: a place where people can play

import UIKit

var str = "Hello, playground"                                      "Hello, playground"


// Declare pi (implicitly Double)
//
var pi = 3.141592                                                  3.141592

// Print pi. Notice we don't use '+'
// to concatenate strings.
//
print ("PI is \(pi))")                                             "PI is 3.141592)\n"

// Declare a radius of 2 (explicitly Double)
// We want to multiply r with pi, and Swift
// is unable to multiply data with different
// types, so we must make sure r is Double
//
var r : Double = 2                                                 2

// Notice how we can write an expression within the double quotations
// in the print statement?
//
print ("Area of circle with radius \(r) = \(pi * r * r)")         "Area of circle with radius 2.0 = 12.5...
```

3. An output panel should appear below. If you are unable to see any output, click on the Play button:



```
// Declare pi (implicitly Double)
//
var pi = 3.141592

// Print pi. Notice we don't use '+'
// to concatenate strings.
//
print ("PI is \(pi))")

// Declare a radius of 2 (explicitly Double)
// We want to multiply r with pi, and Swift
// is unable to multiply data with different
// types, so we must make sure r is Double
//
var r : Double = 2

// Notice how we can write an expression within the double quotations
// in the print statement?
//
print ("Area of circle with radius \(r) = \(pi * r * r)")
```

```
3.141592
"PI is 3.141592)\n"
2
"Area of circle with radius 2.0 = 12.5...
```

```
PI [illegible]592)
Area of circle with radius 2.0 = 12.566368
```

4. Next, we will try to type-cast Doubles / Ints to string, and vice-versa. This is generally useful when you are accepting input from the user, and displaying data on the screen later on.

```
var r1 = "4.50"
var r2 = "6.80a"

// We will try to cast the strings into
// a Double variables.
//
// If the casting of
// the string fails because the string
// is not a valid decimal number, then
// the value of the variable will become nil.
//
// This also means that the type of radius1
// and radius2 are Double? (optionals)
//
var radius1 = Double(r1)
var radius2 = Double(r2)

if radius1 != nil && radius2 != nil
{
    // See how we use "!" to unwrap the value
    // of the optionals to retrieve the underlying
    // result.
    //
    print ("Total radius: \(radius1! + radius2!)")
}
```

5. The code is unable to print the Total Radius because the String r2 cannot be converted to a valid Double. Correct the code to resolve the problem.

6. Now we are going to compute the area of the circle and print it out to the screen, formatting to 2 decimal places. To do so, add the highlighted code below into the if statement in your playground code.

```
if radius1 != nil && radius2 != nil
{
    // See how we use "!" to unwrap the value
    // of the optionals to retrieve the underlying
    // result.
    //
    print ("Total radius: \(radius1! + radius2!)")

    // Now we compute the area using the total radius.
    //
    var totalArea = (radius1! + radius2!) * pi * pi

    // Format the totalArea two 2 decimal places.
    // Notice the external parameter "format:" for
    // the NSString constructor?
    //
    var output = String(format: "%0.2f", totalArea)

    // Print the output to the screen.
    //
    print ("Total area = \(output)")

}
```

Note: We use %0.2f to format the output. To find out more about format specifiers, visit:

http://alvinalexander.com/programming/printf-format-cheat-sheet

7. Run the code to see the output.

Optional:

8. Try creating Int, Bool variables and write code to read / write into those variables.

Strings

In this section, we will run through some basic string operations.

1. Save your previous work, and create a new Playground and call it **MyPlayground1b**.
2. Append the following code to extract a single character, or a range of characters from a given string.

```
import Foundation

var longString = "SUPERcaliFRAGILISTICexpialidocious"
```

```
// Get the character at the n-th position from
// the start and print it.
//
var n = 5
print (longString[
  longString.index(longString.startIndex, offsetBy: n)])

// Get the character at the n-th position from
// the end and print it.
//
n = 6
print (longString[
  longString.index(longString.startIndex, offsetBy: n)])

// Get the substring from position 5 (incl) to 10 (excl)
//
var position5 = longString.index(
  longString.startIndex, offsetBy: 5)
var position10 = longString.index(
  longString.startIndex, offsetBy: 10)
print (longString[position5 ..< position10])
```

3. Next, append the following code to your Playground to convert a whole string to upper-case and lower-case.

```
// Let's convert the entire string to uppercase and
// lowercase and print it out.
//
var lower = longString.lowercased()
var upper = longString.uppercased()
print ("String converted to lower case: \(lower)")
print ("String converted to upper case: \(upper)")
```

4. Finally, append the following code to write a loop to loop through each character of the string, to generate a new string, but with all alternate characters only, and in reversed order.

```
// Then we loop through each character in the string
// and print out its alternate characters in reverse.
//
var reversedString = ""
var count = 0
for c in longString.characters
{
    if count % 2 == 0
    {
        reversedString = String(c) + reversedString
    }
    count = count + 1
}
print ("Reversed string with alt chars only \(reversedString)")
```

5. Run the code to see the output.

Optional:

6. Try to write a loop to count the number of vowels (a, e, I, o, u) and consonants (non-a, e, I, o, u) characters inside a string. Ensure that you take care of upper / lower case characters.

Arrays and Dictionaries

In this section, we will run through some basic array operations.

1. Save your previous work, and create a new Playground and call it **MyPlayground1c**.
2. Append the following code into your Playground to declare, append, remove, and iterate through arrays.

```
import Foundation

// Declare an array of Ints is easy:
var intArray : [Int] = [10, 15, 16, 18, 30]

// Appends an item at the end of the array
intArray.append(40)

// Inserts an item at index 1.
intArray.insert(5, at: 0)

// Removes the element 18 from array, if it exists.
var indexOf18 = intArray.index(of: 18)
if indexOf18 != nil
{
    intArray.remove(at: indexOf18!)
}

// Now loop through each item in the array to print it
for var i in 0 ... intArray.count - 1
{
    print (intArray[i])
}

// Or, you can also do this
for integer in intArray
{
    print (integer)
}
```

3. Append the following code to create, access and print a dictionary.

```
// Declare an associative array (dictionary) of
// string : string pair.
var stocks : [String: String] =
    ["YHOO": "Yahoo! Inc.",
    "CSCO" : "Cisco Systems, Inc."]
```

```
// Adds more stock codes
//
stocks["GOOG"] = "Google Inc."
stocks["AAPL"] = "Apple Inc."
stocks["MSFT"] = "Microsoft Corporation"

// Remove stock codes
stocks.removeValue(forKey: "YHOO")

print ("There are \(stocks.count) stocks")
for (stockCode, company) in stocks
{
    print ("Stock Code: \(stockCode), Company: \(company)")
}
```

4. Run the code to see the output.

Optional:

5. Try to use a [String: Double] to create a dictionary where the key is the student's name, and the value is the student's test marks. Then, print out all students' names and marks, followed by an average of the marks after that.

Functions

In this section, we will write code to declare and call functions.

1. Save your previous work, and create a new Playground and call it **MyPlayground1d**.
2. Append the following code to create, access and print a dictionary.

```
// Declares the function. Take note of
// all the internal / external parameter
// names and the return type.
func sum(theFirst n: Int, numbersOfThisList list: [Int]) -> Int
{
    var sum = 0
    for i in 0 ... list.count - 1
    {
        if (i>=n)
        {
            break
        }
        sum += list[i]
    }
    return sum
}


// We declare a list of integers.
// Notice how we do not declare the type?
// Swift is able to infer the array of Ints type.
var listOfIntegers = [1, 2, 3, 4, 5, 6, 8, 10, 12, 14]

// Call the function.
// Notice the first and second parameter requires
// external parameter names?
```

```
print (sum(theFirst: 4, numbersOfThisList:listOfIntegers))
```

3. Run the code to see the output.

Optional:

4. Try writing a function that swaps two Int or Double variables, then pass in 2 variables to the function and print the output the see the swap taking place.

Classes, Inheritance and Polymorphism

In this section, we will write code to create classes.

1. Save your previous work, and create a new Playground and call it **MyPlayground1e**.
2. Append the following code to declare a Shape class that all other subsequent classes will inherit from.

```
import Foundation

// Declares a shape class.
class Shape
{
    private var name : String

    init (withName: String)
    {
        self.name = withName
    }

    // This method is meant to be overriden
    // by any sub-classes.
    //
    func draw()
    {
        print ("This is a \(name)")
    }
}
```

3. Append the following code to declare a Square class that inherits from the Shape class.

```
// The Square class will inherit the Shape
// class.
class Square : Shape
{
    private var width: Int

    // Initializer
    init(withWidth: Int)
    {
        self.width = withWidth
        super.init(withName: "Square")
    }

    // This class will implement the draw method
    // to draw itself.
    override func draw() {
        super.draw()
        var s = ""
        for i in 0 ... width - 1
        {
            s += "*";
        }

        for i in 0 ... width - 1
        {
            print (s)
        }
    }
}
```

4. Append the following code to declare a Rectangle class that inherits from the Shape class.

```
class Rectangle : Shape
{
    private var width: Int
    private var height: Int

    // Initializer
    init(withWidth: Int, andHeight:Int)
    {
        self.width = withWidth
        self.height = andHeight
        super.init(withName: "Rectangle")
    }

    // This class will implement the draw method
    // to draw itself. Look at how it is different
    // from the Square class?
    override func draw() {
        super.draw()
        var s = ""
        for i in 0 ... width - 1
        {
            s += "*";
        }

        for i in 0 ... height - 1
        {
            print (s)
        }
    }
}
```

5. Then we declare a list of Shapes as an array, append some Square and Rectangle objects and see how polymorphism kicks into action.

```
// Declare an array of Shapes
```

```
//
var shapes : [Shape] = []

// Add 2 squares and rectangles
//
shapes.append(Square(withWidth: 2))
shapes.append(Square(withWidth: 3))
shapes.append(Rectangle(withWidth: 1, andHeight: 4))
shapes.append(Rectangle(withWidth: 5, andHeight: 2))

// Loop through each shape and use polymorphism
// to call their respective draw methods.
//
for shape in shapes
{
    shape.draw()
}
```

6. Run the code to see the output.

Challenge:

1. Try writing a Circle class that takes in a radius during initialization, and overrides the draw method to print a circle (consisting of '*' character) to the output.

Swift's Optionals

(This Section is Required, In Case You are Wondering)

In this section, we will write code to declare and use optional variables.

1. Create a new Playground and call it **MyPlayground2a**.
2. We will create two classes here, Teacher and Student. Append the following code to your Playground:

```
// Declare a student class
class Student
{
    var name: String

    // Declare implicitly unwrapped optionals
    var mathsTeacher: Teacher! = nil
    var scienceTeacher: Teacher! = nil

    // Initialize the student with a name and
    // teachers. Notice that the parameters
    // are non-implicitly unwrapped optionals.
    init(name: String, mathsTeacher:Teacher?,
        scienceTeacher: Teacher?)
    {
        self.name = name

        // Notice how you can still assign the parameter
        // to the class variable, even though the former
        // is Teacher? type and the latter is Teacher!
        // type
        self.mathsTeacher = mathsTeacher
        self.scienceTeacher = scienceTeacher
    }

}


// Declare a teacher class
class Teacher
{
    var name: String

    // Declare the teacher's age as optional
    // but it needs to be explicitly unwrapped
    // to get its value
    var age: Int?

    // Initialize the teacher with a name and optional age
    init(name: String, age: Int?)
    {
        self.name = name
        self.age = age
    }
}
```

3. Modify the following highlighted code inside the Teacher class to add a method to print information about the Teacher.

```
class Teacher
{
    // Let's print information about this teacher.
    func printTeacherInfo()
    {
        print ("    Name: \(name)")
        if (age != nil)
        {
            // Notice how we unwrap age to retrieve the actual value?
            print ("    Age:  \(age!)")
        }
        else
        {
            print ("    Age:  SECRET!")

        }
    }
}
```

4. Modify the following highlighted code inside the Student class to add a method to print information about the Student.

```
class Student
{
    // Print information about the student and his/her teachers.
    func printStudentInfo()
    {
        print ("Name: \(name)");

        if (mathsTeacher != nil)
        {
            print ("  Maths Teacher:")
            // Notice how we do not need to unwrap with a '!'?
            mathsTeacher.printTeacherInfo()
        }
        if (scienceTeacher != nil)
        {
            print ("  Science Teacher:")
            // Notice how we do not need to unwrap with a '!'?
            scienceTeacher.printTeacherInfo()
        }
    }
}
```

5. Append the following code to declare some teachers and students, and print out the list of students and their respective teacher's information.

```
var mrEinstein = Teacher(name: "Albert Einstein", age: 24)
var mrNewton = Teacher(name: "Sir Isaac Newton", age: nil)
var mrBabbage = Teacher(name: "Charles Babbage", age: 30)
var mrPascal = Teacher(name: "Blaise Pascal", age: 29)

var students = [
    Student(name: "Peter", mathsTeacher: mrPascal,
        scienceTeacher: nil),
    Student(name: "Mary", mathsTeacher: nil,
        scienceTeacher: mrEinstein),
    Student(name: "Tom", mathsTeacher: mrBabbage,
        scienceTeacher: mrNewton)
]

for student in students
{
    student.printStudentInfo()
}
```

6. Run the code to see the output.

Optional:

7. Try to remove the checks for nil and see what happens when you try to access the nil-valued optional.