CPSC 411 Spring 2020

This study guide is for the CPSC-411 Midterm for Fall 2020, given next Monday, 16 March at 1900 hours. The midterm will last the entire class time, to give students sufficient time for the exam. I will be online to answer any questions you may have.

Note: the midterm is timed, and will start exactly at 1900 hours this coming Monday, 16 March 2020, and will end at 2145 hours. If you are late, you cannot have extra time for the exam.

The midterm is composed mostly of programming questions from Swift and Swift UI. You will use a combination of a SwiftUI project and a Swift project to write and test your code.

As you check off each problem to be completed, you can add it to a text file that you will submit to Titanium when the midterm is complete.

Example:
SwiftUI Question #1: Write a SwiftUI program that has a button in the center, and shows how many times you have tapped it. After creating and testing the SwiftUI code to do this, you would add that to the SwiftUI (top) portion of your test submission.
SwiftUI Question #2. Write a SwiftUI program that shows "Hello, world!" three times in succession. You would clear the previous project, write and test the code for question #2, and add its code to the SwiftUI (top) portion of your midterm submission, and so on.

Swift Question #1: Create an instance of an immutable Swift string, and print it out. You would write this code in a Swift project, and test it, and add it to the Swift (bottom) section of your midterm.
Swift Question #2: Create and instance of a mutable String, and print it out, and prove that it is mutable, or state that it is not, and give the reason why. Append your code to do this to your Swift playground, test it, and append it to the Swift section of your midterm submission, and so on.

In the end you will have:
SWIFT UI SECTION
#1 SwiftUI Answer
#2 SwiftUI Answer
…

SWIFT SECTION
#1 Swift Answer
#2 Swift Answer
…

END of midterm submission

You will need a Swift and SwiftUI compiler to take the midterm online, using a combination of a SwiftUI project (for the SwiftUI section), and a Swift Playground (for the Swift section).

In this midterm, you will demonstrate your basic and intermediate knowledge of both the SwiftUI and Swift language.
Typical questions and solutions are provided for both of these sections. Actual test questions may vary.

The SwiftUI questions will be very similar to the first three quizzes you have taken. The solutions to these quizzes are

provided within this document. **Typical Swift questions and answers are also provided within this document. Actual test questions may vary.**

The SwiftUI programming section includes questions on how to create basic UI interfaces using Swift UI. If you have been following along with the HackingWithSwiftUI videos, you should be in good hands.

The Swift programming section includes questions on basic data types (Bool, Int, Double, and String), String processing, mutable vs. immutable variables, lists, tuples, and dictionaries, conditionals, loops, and ranges. You will also be tested on arrays, optionals, closures, functions, structs and classes.

Finally, you will be asked to write a short app, using the knowledge from chapters 1 through 14 (excluding 7,8, and 9). The app will be something you can code in about 60 minutes. It should be similar in difficulty to the TicTacToe app you created previously (see the following example: https://learnappmaking.com/creating-a-simple-ios-game-with-swift-in-xcode/).

The following sources will be helpful for you in reviewing Swift:
(from the Safari online database for Swift): LEARNING SWIFT, by Manning, Butterfield-Addison, and Nugent, chapters 2 and 3: The Basics of Swift, and Swift for Object-Oriented App Development)
(SWIFT PROGRAMMING, The Big Nerd Ranch Guide, by Mathias and Gallagher, part II: The Basics, part III: Collections and Functions, and Ch. 15: Structs and Classes)

Types, Constants,Variables, Conditionals, Numbers, Switches, Loops, and Strings

Create an instance of an immutable String, and print it out.
Create an instance of a mutable String, and print it out, then prove that it is.
Print out all of the characters in the following String: “abcdefghijklmnopqrstuvwxyz”, one by one, using a for loop. Write out a formula in Swift for printing out the 6th character in the String. Hint: You will need an Index to do that. Write out a formula in Swift (in one line) for finding the length of the String: “Call me Ishmael.”
Print out the unicode values (base 10) of the above String “Call me Ishmael.”
Print out the unicode values (base 16) of the above String “Call me Ishmael.”

`Hint: print(String(format:"0x%02x", arguments:[c.value]), terminator: " ");`

Write out the formula for reversing the String. Create a String equivalent of the number 2371.

Given that “middle” in Chinese is “zhong1”: unicode 4e2d, and “kingdom” is: “guo2”: 570d, print the Chinese characters for "China (zhong1 guo2)" in unicode.

Convert the following String to all uppercase: “`cal state fullerton`”.
Change all of the vowels (a,e,i,o,u, and y) to be ‘Z’.
Change all instances of “`ss`” in the following string to be ‘`AMTRAK`’: “`Mississippi`”.

Determine if the following String has the prefix or suffix “__”: __add__.
For the above String (“__add__”), strip the prefix/suffix “__” to get just: “add”. Using a range, print out all of the cubes (x^3) of x from x=1 to 10, inclusive. Using a range, print out all of the squares of x from x=[1,10).
Write an instance of a variable of type Int that declares the type and its value. Write an Int instance that declares its value but not its type.
Write an instance of an Int more than 10000000 in value that uses underscores.

Create a switch statement that handles the following ranges of error codes: 100 or 101: Informational, 1xx”
204: “Successful but no content, 204”
300 to 307: “Redirection: 3xx”

400 to 505: “Server error”
otherwise: “Unknown error. Please retry”

Create a named tuple: lname, fname, cwid, gpa that describes a student, using a typealias similar to this: typealias point = (x: Double, y: Double). Then, create a student, and print out the tuple's elements from the student

if you have the following code: let point = (x: 1, y: 4), write a switch that uses cases and where statements to determine if the angle between the point and the x axis is acute, right, obtuse, or straight.

Write a Switch statement that prints out "I will study Swift every day" 25 10 times. Be sure that each statement is numbered, and that the lines are all right-justified.

1. I will study Swift every day. 2. I will study Swift every day. ...

Print out all of the multiples of 3 from 1 to 30 using a Range and a where statement. Print out on the same line the tuple of 16, Mar, 2020, w/ each field separated by "/".

Write a function that prints out a table of conversions from Fahrenheit to Celsius, from -50 F to 150 F, in steps of 10 degrees Fahrenheit. Each line should have the Fahrenheit temperature and the corresponding Celsius temperature. Write the program using functions.

Optionals, Arrays, Dictionaries, Sets, Closures, Structs, Classes, and Functions

Write a function that that prints a greeting, but returns a Void. Write a function that takes two ints, and returns the sum.

Write a function that converts Fahrenheit temperatures to Celsius (C = 5/9.0 * (F – 32).

Write a function that takes a list of ints, and returns a list of numbers that are multiples of 3.

Write a function that takes a variable number of integers, and returns a tuple containing (minimum: Int, maximum: Int).

Create an Optional String variable that contains the "666". Write the code that optionally unwraps that, and if successful, create an optional Int that converts it to an Int.

Create a Set that contains all items from both Sets.
Create a Set that contains all of the items that are not in common.
Create a Dictionary of 3 US states, and 2 cities from each state. Then, print out all of the cities from the Dictionary.

Make an Array of seven Strings, and search for a String in that Array. Write code that will print out the index of that String if found, or that prints out "nil" if it is not.

Make an array of several Strings, and IN ONE LINE, write the code that prints them out in reverse, each on its own line.

Write a function called siftBeans(fromGroceryList:) that takes a grocery list (as an array of strings) and "sifts out" the beans from the other groceries. The function should take one argument that has a parameter name called list, and it should return a named tuple of the type (beans: [String], otherGroceries: [String]).

Here is an example of how you to call your function and what the result should be:

```
let result = siftBeans(fromGroceryList: ["green beans", "milk", "black beans", "pinto beans", "apples"])
```

Here is an example of how you should be able to call your function and its result

```
// output

// result.beans == ["green beans", "black beans", "pinto beans"] // true
// result.otherGroceries == ["milk", "apples"] // true
```

Hint: You may need to use a function on the String type called hasSuffix(_:).

Write a function that divides a list of words into those before a given dividing String, and those equal or after to that String.

```
let divided = partitionStrings(["at", "be", "cat", "dog", "fox", "lima", "snake", "yak"],
```

```
print(divided)
```

// output

```
byDivisor: "goat")

// (before: ["at", "be", "cat", "dog", "fox"], equalAndAfter: ["lima", "snake", "yak"])
```

Hint: You may find the following function helpful

```
func sortedEvenOddNumbers(_ numbers: [Int]) -> (evens: [Int], odds: [Int]) { var evens =
[Int]()
var odds = [Int]()
for number in numbers {

if number % 2 == 0 { evens.append(number); } else { odds.append(number)
}

}

    return (evens, odds)
}
```

FROM SWIFT UI QUIZ ONE

1. **(1 pt)** In SwiftUI, write the code to show “Hello, world!”
```
struct ContentView: View {
    var body: some View { Text(“Hello, world!”) }
 }
```
2. **(1 pt)** In SwiftUI, write the code to show “Hello, world!” twice
```
struct ContentView: View {
    var body: some View {
       VStack { Text(“Hello, world!”); Text(“Hello, world!”) }
    }
 }
```
3. **(1 pt)** In SwiftUI, write the code to show a form with “Hello, world!” written twice in one section, and one in another)
```
struct ContentView: View {
  var body: some View{
    Form {
      Section {
        Text(“Hello, world!”);
        Text(“Hello, world!”)
      }
      Section {
        Text(“Hello, world!”)
      }
    }
  }
}
```
4. **(1 pt)** In SwiftUI, write the code to show a form with “Hello, world” shown twice, and with with a Navigation bar, and the text “SwiftUI” shown in its title.
```
struct ContentView: View {
  var body: some View{
    NavigationBar {
      Form {
        Section {
          Text(“Hello, world!”);
          Text(“Hello, world!”)
        }
        Section { Text(“Hello, world!”) }
      }.navigationBarTitle(“SwiftUI”)
    }
  }
}
```
5. **(1 pt)** In UIKit, views are built with **subclasses of class UIView**,

   but in SwiftUI, they are built with **subclasses of struct View**.

6. **(2 pts)** Write the SwiftUI code to have a button that shows its tapCount when tapped on.

```
struct ContentView: View {
  var body: some View{
    Button(action: { self.tapCount += 1}) {
      Text("Tap count: \(self.tapCount)")
        .padding()
        .frame(width: 100, height: 100)
        .background(Color.blue)
        .foregroundColor(.white)
        .multilineTextAlignment(.center)
      }
    }
  }
}
```

7. **(2 pts)** Write the SwiftUI code that has a form with a TextField that asks the user to enter their name, and a Text object that says, "Your name is _____" (show name).

```
struct ContentView: View {
  @State private var name = ""
  var body: some View {
    TextField("Enter your name: ", text: $name)
    Text("Your name is: \(name)")
  }
}
```

8. **(2 pts)** Write a SwiftUI form that shows 100 rows (0 – 99), with each rows showing the word "Row" and its number.

```
struct ContentView: View {
  var body: some View{
    Form {
      ForEach(0 ..< 100) {
        Text("Row \($0)")
      }
    }
  }
}
```

9. **(1 pt)** Fill in the missing code that has a Picker to select from one of the following three students:

```
struct ContentView: View {
  let students = ["Harry", "Hermione", "Ron"]
  @State private var currentStudent = "Harry"

  var body: some View{
    VStack {
      Picker("Select your student", selection: $currentStudent) {
        ForEach(0 ..< students.count) {
          Text(self.students[$0])
        }
      }
    }
  }
```

FROM SWIFT UI QUIZ TWO

10. **(1 pt)** In SwiftUI, what element do you use to get three Text's containing "X" stack vertically?
    VStack

11. **(1 pt)** In SwiftUI, what do you use to get three Texts containing "O" stack next to each other?
    Hstack

12. **(1 pt)** In SwiftUI, how do you get Texts to be on top of each other (one hiding the other)?
    ZStack

13. **(1 pt)** In SwiftUI, write the code to show text (your choice) with a blue-green background covering it.

```
struct ContentView: View {
  var body: some View {
    Text("victory goes to the swift").padding().frame(width:100, height: 100)
      .background(Color(red: 0, green: 1, blue: 1))
  }
}
```

14. **(1 pt)** In SwiftUI, write the code to show a linear gradient from white to black, from top to bottom.

```
struct ContentView: View {
    var body: some View{
            Text("Hello")
      .padding()
      .padding().frame(width:200, height: 200)
      .background(LinearGradient(gradient: Gradient(colors: [.white, .black]),
                  startPoint: .top, endPoint: .bottom))
   }
 }
```

15. **(1 pt)** In SwiftUI, write the code to have a button with the image of a pencil next to the word Edit, which prints "Tapped" when the button is tapped. (hint: use systemName: "pencil")

```
struct ContentView: View {
  var body: some View{
    Button(action: { print("Tapped") }) {
      HStack {
        Image(systemName: "pencil")
        Text("Edit")
      }
    }
  }
}
```

16. **(2 pts)** Write the SwiftUI code to have a button that shows its tapCount when tapped on.

```
struct ContentView: View {
  @State private var showingAlert = false
  var body: some View {
    Button("Show Alert: ") {
      self.tapCount += 1
      self.showingAlert = true
    }
    .alert(isPresented: $showingAlert) {
      Alert(title: Text("Tapcount"), message: Text("\(self.tapCount)"),
            dismissButton: .default(Text("OK")))
    }
  }
}
```

17. **(2 pts)** Fill in the missing SwiftUI code to randomize the countries, space out the text and the flags by 30, and to color the entire background of the iOS device by blue

```
struct ContentView: View {
  @State private var name = “”
  @State private var countries = [“Germany”, “China”, “USA”].shuffled()
  @State private var correctAnswer = Int.random(in: 0...2)

  var body: some View{
    ZStack {
      LinearGradient(gradient: Gradient(colors: [.blue, .black]),
                     startPoint: .top,
                     endPoint: .bottom).edgesIgnoringSafeArea(.all)
      VStack(spacing: 30) {

        VStack { Text(“Tap flag of”); Text(countries[correctAnswer]) }
          ForEach(0..<3) { number in
            Button(action: {
                    // flag tapped
            }) { Image(self.countries[number]).renderingMode(.original) }
          }
        }
      }
    }
  }
```

18. **(2 pts)** Write a SwiftUI form that shows 100 rows (0 – 99), with each rows showing the word “Row” and its number.

```
struct ContentView: View {
  var body: some View {
    Form {
      ForEach(0 ..< 100) {
        Text(“Row \($0)”)
      }
    }
  }
}
```

19. **(1 pt)** Write an example of a Swift closure called walking, that simply prints out, “I am going for a walk”, (it takes no parameters), and call it like this: `walking()`.

```
let walking = {
   print(“I’m going for a walk”) }
}
```

20. **(1 pt)** Write an example of a Swift closure called driving, that simply prints out “I am going to _____”, that takes a parameter. It will be called like this: `driving(“London”)`

```
let travel = {(place: String) in
  print("I am  {going to \(place)")
}
```

21. **(1 pt)** Write one shorthand versions of the following closure (assuming that Swift already knows that it takes and returns a String from its usages):

```
  let travel = { $0 } { print("I am going to " + $0) }
    travel("Chicago")
```

FROM SWIFT UI QUIZ THREE

1. **(1 pt)** According to Apple's documentation for UIKit, about how many methods and variables are in UIView?
   Approximately 200

2. **(1 pt)** In SwiftUI, how many methods and variables are in View (from which ContentView derives?
   One

3. **(1 pt)** Choose the code fragments below that are valid SwiftUI code (there may be none).
   var body: some View { Text("Hello, world!") }
   var body: Text { Text("Hello, world!") }
   ```
   var body: Button { Button(action: { print("Tapped") }) { Text("Tap") }}
   ```

4. **(1 pt)** In SwiftUI, write the code to show a Text (your choice) with a red background completely behind it, including the entire safe area?
```
struct ContentView: View {
  var body: some View {
    Text("Hello, world!")
      .frame(maxWidth: .infinity, maxHeight: .infinity)
      .background(Color.red)
      .edgesIgnoringSafeAreaSafeArea(.all)
  }
}
```

5. **(1 pt)** In SwiftUI, write the code that would show the view to the right when running.
```
struct ContentView: View {
  var body: some View {
     Button("Hello, world!")
      .frame(width: 200, height: 200)
      .background(Color.red)
  }
}
```

6. **(1 pt)** In SwiftUI, write the code that would show the view to the right when running.
```
struct ContentView: View {
  var body: some View {
    Text("Hi!")
      .padding()
      .background(Color.red)
      .padding()
      .background(Color.blue)
      .padding()
      .padding()
      .background(Color.green)
  }
}
```

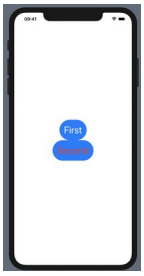
7. **(1 pt)** In SwiftUI, write the code that would show the view to the right when running.

```
struct ContentView: View {
  var body: some View {
    VStack {
      Text("Gryffindor").font(.largeTitle)
      Text("Hufflepuff")
      Text("Ravenclaw")
      Text("Slytherin")
    }
  }
}
```

8. **(1 pt)** In SwiftUI, write the code that would show the view to the right when running.

```
struct CustomView: View {
  var text: String
    var foregroundColor: Color

    var body: some View {
    Text(text)
      .font(.largeTitle)
      .padding()
      .background(Color.blue)
      .clipShape(Capsule())
  }
}
struct ContentView: View {
  var body: some View {
    VStack(spacing: 10) {
      CapsuleText(text: "First")
        .foregroundColor(.white)
      CapsuleText(text: "Second")
        .foregroundColor(.yellow)
  }
}
```



9. **(1 pt)** In SwiftUI,

```
struct Title: ViewModifier {
  func body(content: Content) → some View {
    content
      .font(.largeTitle)
      .foregroundColor(.white)
      .padding()
      .background(Color.blue)
      .clipShape(RoundedRectangle(cornerRadius: 10))
  }
}

extension View {
  func titleStyle() -> some View {
    self.modifier(Title())
  }
}

struct ContentView: View {
  var body: some View{ Text('Hello!').titleStyle() }
}
```

10. write the code to show a linear gradient from white to black, from top to bottom.

```
struct ContentView: View {
  var body: some View{
    Text("Hello")
      .padding()
      .padding().frame(width:200, height: 200)
      .background(LinearGradient(gradient: Gradient(colors: [.white, .black]),
                  startPoint: .top, endPoint: .bottom))
  }
}

      // OR
struct ContentView: View {
  var body: some View{
    LinearGradient(gradient: Gradient(colors: [.white, .black]),
                  startPoint: .top, endPoint: .bottom)
  }
}
```