Lab Quiz 5 Template

TA Instructions

Before Lab:

- Copy this template into a new file and put it in the Lab Quiz 5 folder with your lab section in the name. Do not edit this template directly since other TAs will need to copy this
- Fill in all the personalizations for each question template **before your lab section**
    - Feel free to alter the questions beyond the templates as long as you're testing the same concepts. The template provides a minimal amount of variance between labs
- **Run your code** to be 100% sure of the answers and that there are no typos in the code you give the students
- Have a plan to display the quiz to your students
    - Setup a github repo and have students view the code on their computers with close supervision to ensure they only access the quiz code

During Lab:

- Give them 1 hour to complete the Quiz. You don't have to start at the beginning of lab but you must end the quiz 1 hour after you start

Grading:

- Start grading as the quiz ends. When you're done grading, add their scores into AutoLab and release the grades for your section. Their grades should be posted within 24 hours of taking the quiz. You can hand the quizzes back next week

Student Instructions

- This is an exam environment
    - No talking
    - No material allowed except blank paper and writing utensils
    - Only ask TA's questions clarifying what a question is asking

Q1 (30 points): Study the following code to answer the question below

```
case class PassVal(value: Int)

class Actor1(nextActor: ActorRef) extends Actor {
 def receive: Receive = {
   case message: PassVal =>
     if (message.value > 0) {
       nextActor ! PassVal(message.value - 2)
     }
     else if (message.value < 0) {
       nextActor ! PassVal(message.value + 1)
     }
     else {
       nextActor ! PassVal(3)
     }
 }
}

class Actor2() extends Actor {
 def receive: Receive = {
   case message: PassVal =>
     println(message.value)
 }
}


object Q1 {

 def main(args: Array[String]): Unit = {
   val system = ActorSystem("Quiz5")

   val mainActor = system.actorOf(Props(classOf[Actor2]))

   val first = system.actorOf(Props(classOf[Actor1], mainActor))
   val second = system.actorOf(Props(classOf[Actor1], first))
   val third = system.actorOf(Props(classOf[Actor1], second))
   val fourth = system.actorOf(Props(classOf[Actor1], third))
   val fifth = system.actorOf(Props(classOf[Actor1], first))

   fourth ! PassVal(3)
   fifth ! PassVal(0)
 }
}
```

What two values are printed when this program runs (order does NOT matter) ?

Q2 (30 points): Study the following code to answer the question below. You may assume all necessary imports are included.

This TCP Socket Server is running

```
class TCPServer() extends Actor {

 import Tcp._
 import context.system

 IO(Tcp) ! Bind(self, new InetSocketAddress("localhost", 8000))

 var buffer: String = ""
 val delimiter: String = "~"

override def receive: Receive = {
     case b: Bound => println("Listening on port: " + b.localAddress.getPort)
     case c: Connected =>
       sender() ! Register(self)
       sender() ! Write(ByteString("Connected" + delimiter))
     case r: Received =>
       buffer += r.data.utf8String
       while (buffer.contains(delimiter)) {
         val message = buffer.substring(0, buffer.indexOf(delimiter))
         buffer = buffer.substring(buffer.indexOf(delimiter) + 1)
         sender() ! Write(ByteString("Message Received" + delimiter))
       }
   }
 }

 object Q2 {
   def main(args: Array[String]): Unit = {
     val actorSystem = ActorSystem()
     actorSystem.actorOf(Props(classOf[TCPServer]))
   }

}
```

Then this TCP Socket Client is ran

```
scala_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
scala_socket.connect(('localhost', 8000))

delimiter = "~"
number = [0]

def listen_to_scala(the_socket):
   buffer = ""
   while True:
       buffer += the_socket.recv(1024).decode()
       while delimiter in buffer:
           message = buffer[:buffer.find(delimiter)]
           buffer = buffer[buffer.find(delimiter) + 1:]
           get_from_scala(message)


def get_from_scala(message):
   if "Connected" in message:
       send_to_scala("Hello Scala!")
   else:
       number[0] += 1
       print(number[0])
```

```python
def send_to_scala(data):
   scala_socket.sendall((json.dumps(data) + delimiter).encode())


send_to_scala("Hello!")
listen_to_scala(scala_socket)
```

What is the final value of number[0]?

Q3 (40 points): Study the following code containing a subset of the Clicker 2 functionality.

`server.py`

```python
from flask import Flask, request, send_from_directory
from flask_socketio import SocketIO

app = Flask(__name__)
socket_server = SocketIO(app)

sidToUsername = {}
clicks = {}


@socket_server.on('register')
def register(username):
   sidToUsername[request.sid] = username
   if username not in clicks:
       clicks[username] = 0
   socket_server.emit("message", str(clicks[username]), room=request.sid)
   print(username + " connected")


@socket_server.on('disconnect')
def disconnect():
   if request.sid in sidToUsername:
       username = sidToUsername[request.sid]
       del sidToUsername[request.sid]
       print(username + " disconnected")


@socket_server.on('clickGold')
def click_gold():
   username = sidToUsername[request.sid]
   socket_server.emit("message", str(clicks[username]), room=request.sid)
   print(clicks)


@app.route('/')
def index():
   return send_from_directory('.', 'game.html')


@app.route('/<path:filename>')
def static_files(filename):
   return send_from_directory('.', filename)


print("Listening on port 8080")
socket_server.run(app, port=8080)
```

`game.html`

```html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Clicker Quiz</title>
   <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
</head>
<body>
```

```
<button id="gold" onclick="clickGold();">GOLD!</button>
<div id="displayGold"></div>
<script src="game.js"></script>

</body>
</html>
```

game.js

```
var socket = io.connect({transports: ['websocket']});

socket.on('message', function (event) {
   document.getElementById("displayGold").innerHTML = event;
});

function clickGold(){
   socket.emit("clickGold");
}
```

ScalaClient.scala

```
import io.socket.client.{IO, Socket}
import io.socket.emitter.Emitter


class HandleMessagesFromPython() extends Emitter.Listener {
 override def call(objects: Object*): Unit = {
   val gold = objects.apply(0).toString
   println("I have " + gold + " gold")
 }
}


object ScalaClient {

 def main(args: Array[String]): Unit = {

   val socket: Socket = IO.socket("http://localhost:8080/")
   socket.on("message", new HandleMessagesFromPython)

   socket.connect()
   socket.emit("register", "ScalaUser")

   socket.emit("clickGold")
   socket.emit("clickGold")
   socket.emit("clickGold")
   socket.emit("clickGold")
   socket.emit("clickGold")
 }

}
```

When this code is working properly it should have the following features:

- When server.py is ran it hosts game.html/js and listens for websocket connections on port 8080
- When a user sends a register message to the python server with a username it will associate this username with their socket id and setup a data structure to remember the number of clicks they've made

- If a username registers again they continue with the same number of clicks they've had (The server effectively saves their game, though it does not use persistent storage so saved games are lost when the server restarts)
- When a web client connects they see a page with a gold button and a display of their current gold (number of times they've clicked the button). As they click, the number is incremented. If they reconnect they see their total number of clicks across all connections
- Each time the scala client is ran it will simulate 5 clicks of a gold button and print out the total clicks from Scala after each click

The code provided does not fully realize all of these features. There are 2 bugs in the code. Find each bug, describe which feature is broken by the bug, and explain why the feature is broken, and how you'd fix it

Bug #1:

Where is the bug?

What feature does this break?

How is the feature broken?

What would you do to fix the bug?

Bug #2:

Where is the bug?

What feature does this break?

How is the feature broken?

What would you do to fix the bug?