Lab exercise 1 – Sam da Costa (p11469sd)

Task 1.1

URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php**
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:0:{}

URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php?action=set&key=nurse1&value=hello**
Result: {blank page}
URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php**
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:1:{s:6:"nurse1";s:5:"hello";}

URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php?action=set&key=key1&value=value1**
Result: {blank page}
URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php?action=set&key=key2&value=value2**
Result: {black page}
URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php**
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:3:{s:6:"nurse1";s:5:"hello";s:4:"key1";s:6:"value1";s:4:"key2";s:6:"value2";}

URL: **https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php?action=unset&key=nurse1**
Result: {blank page}
URL: https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:2:{s:4:"key1";s:6:"value1";s:4:"key2";s:6:"value2";}

Task 1.2

```
sam@SamPCLinux:~/GitRepos/comp28112_ex1$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import im
>>> server = im.IMServerProxy("https://web.cs.manchester.ac.uk/p11469sd/comp2811
2_ex1/IMserver.php")
>>> server['pythonTest1'] = "hello"
>>>
```

URL:
https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:3:{s:4:"key1";s:6:"value1";s:4:"key2";s:6:"value2";s:11:"pythonTest1";s:5:"hello";}

```
>>> print(server['pythonTest1'])
b'hello\n'
>>> del server['pythonTest1']
>>>
```

URL:
https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:2:{s:4:"key1";s:6:"value1";s:4:"key2";s:6:"value2";}

```
>>> server['setviapy'] = 'vartoset'
>>>
```

URL:
https://web.cs.manchester.ac.uk/p11469sd/comp28112_ex1/IMserver.php
Result:

COMP28112 Server: Messaging System for Healthcare Professionals

a:3:{s:4:"key1";s:6:"value1";s:4:"key2";s:6:"value2";s:8:"setviapy";s:8:"vartoset";}

Task 1.3
Question 1

My general strategy for this exercise will be to focus on getting the two users on startup to an agreed upon state. Once I have this state agreed, creating the back and forth between the users should be trivial. Another strategy I used was that I avoided else statements as much as possible. I aimed to do this as I want to avoid as much undefined behaviour as possible. What I mean by this is that I want the program to move from state to state when I see specific values in my control variables. With else statements I can't know exactly what is in the control variables in the else block, whereas if I specify the values in an if statement, I have a guarantee of what they started as.

My protocol is to initialise the first user to the write state, and then initialise the second user to the read state. The user specified in server["control"] will be the user in the write state. When a user has control, they should wait for user input to write, write it and then wait for the other user to read. Once the other user reads, they then take control and the same process occurs for the other user. If at any-point the users crash, the server should be reset to a know position. I managed to implement this so that my program can reset itself on keyboard interrupts. In addition, if the program starts in an unusual state where the server thinks two users are already connected, the server will stop the connection and reset itself.

I chose this protocol structure, as it seemed the simplest and cleanest to implement. There aren't any um-necessary states which the user must progress through. User A will follow the process:

Initialise → wait for input → write → wait for response → output response → go to step 2
and User B will do the reverse:
Initialise → wait for response → output response → wait for input → write → go to step 2

Question 2
A limitation of this system is how users have to wait for a response to respond again. A way to change this would be to have a server to handle the communication between the two. This example program works more like a peer to peer connection in the sense that to communicate properly both clients need to know exactly what the other user is expecting. With a server to act as an intermediary rather than just putting data into a dictionary and expecting the user to know what to do with it, you could have a request based architecture, where you ask the server to store and retrieve data and it responds with whether you are allowed to or not at that time.