Portfolio assignment 1: Bad bots

DATA2410, Spring 2021

- **This is an Individual assignment. It must be your work.**
- The assignment is a part of your portfolio and counts towards your final grade.
- Allowed programming languages are python3, java, or C++[1]. Other languages may be acceptable on request, depending on resources.

Contents:

Introduction

To make a simple chat bot all you need is a function that takes a string as input (what somebody said) and returns another string (the bot's response). A more complex bot would have a longer memory, but responding to a single sentence is more than hard enough so let's stick with that. The main part of this assignment is to make a little chat server with a couple of bad chat bots in it. To make it simpler, let's say that your bots (your functions) never have to take initiative but only respond to what someone else is suggesting. In the simplest case the suggestion is a single verb - a suggestion for something to do to pass the time. E.g. "Let's eat!" or "Wanna play?". How your bots respond to these suggested actions is completely up to you, but you should have 3 different ones.

For this assignment Bots are functions that take a string or two as input and returns one or two strings:

```python
def my_bot(action, alt_action = None):
    return "I think {} sounds great!".format(action + "ing"), None
```

[1] C++ is anything that compiles with a C++ compiler - the C subset of C++ is fine

Example bots

This is a very simple example of how the bots could be implemented:

```
import random

def alice(a, b = None):
    return "I think {} sounds great!".format(a + "ing")

def bob(a, b = None):
    if b is None:
        return "Not sure about {}. Don't I get a choice?".format(a + "ing")
    return "Sure, both {} and {} seems ok to me".format(a, b + "ing")

def dora(a, b = None):
    alternatives = ["coding", "singing", "sleeping", "fighting"]
    b = random.choice(alternatives)
    res = "Yea, {} is an option. Or we could do some {}.".format(a, b)
    return res, b

def chuck(a, b = None):
    action = a + "ing"
    bad_things = ["fighting", "bickering", "yelling", "complaining"]
    good_things  = ["singing", "hugging", "playing", "working"]

    if action in bad_things:
        return "YESS! Time for {}".format(action)
    elif  action in good_things:
        return "What? {} sucks. Not doing that.".format(action)
    return "I don't care!"

action = random.choice(["work", "play", "eat", "cry", "sleep", "fight"])

print("\nMe: Do you guys want to {}? \n".format(action))
print("Alice: {}".format(alice(action)))
print("Bob: {}".format(bob(action)))
print("Dora: {}".format(dora(action)[0]))
print("Chuck: {}".format(chuck(action)))
```

Output:

```
alfredb@Alfreds-MacBook-Pro:~/portfolio1$ python3 example.py
Me: Do you guys want to eat?
Alice: I think eating sounds great!
Bob: Not sure about eating. Don't I get a choice?
Dora: Yea, eat is an option. Or we could do some coding.
Chuck: I don't care!

alfredb@Alfreds-MacBook-Pro:~/portfolio1$ python3 example.py
Me: Do you guys want to fight?
Alice: I think fighting sounds great!
Bob: Not sure about fighting. Don't I get a choice?
Dora: Yea, fight is an option. Or we could do some singing.
Chuck: YESS! Time for fighting
```

You should try to make the bots more interesting. The following is output from an alternative implementation with a little more code:

```
Host:  Let's all cry together today!
Chuck: Awesome! I've been longing for some crying all week!
Bob:   crying seems negative. And I wanted more choices! That's it then? We're crying?
Then let's get to it!💥
Alice: Again with the crying!😩
Dora:  Oh, crying, excellent idea. Could also nag maybe?


Host:  Why don't we steal?
Chuck: So stealing it is then. I don't mind🤓
Bob:   stealing seems horrible. And I wanted more choices! Since nobody seems to have
any better ideas I guess we're stealing🤷
Alice: Are you serious? stealing is the last thing we need.
Dora:  Right, stealing is one alternative I guess. Or we could race.


Host:  Why don't we walk?
Dora:  Meh. I did some walking last night. I'll complain maybe.
Chuck: Are you serious? walking is the last thing we need.
Alice: Somebody suggested walking? Sure, I'm up for anything!
Bob:   walking seems great. And complaining seems lame. Awesome! I've been longing for
some walking all week!
```

Task 1: TCP client

Implement a TCP client program, "client.py", that takes 3 command line parameters: *ip, port,* and *bot.* The idea is that each client can connect from its own terminal and run a single bot each. Several terminal windows can run in parallel, connecting different bots to the same server. You can provide command line parameters if you like, but they must then be explained if you call the program with "--help" or "-h". The client will do the following:

1. Create a TCP socket and connect to *(ip, port)*
2. Read from the socket, line by line.
    a. If the line is from the host, you can expect it to be a suggestion, e.g. "Let's take a walk" or "Why don't we sing?". Extract the suggested action from the line. E.g. "walk" or "sing".
        i. Call the function *bot* to create a response. Send the response back over the socket.
        ii. You can choose to remember the suggested action as alternative 1.
    b. If the line is from one of the other participants you can choose to ignore it, or pass it to your bot as alternative 2 if there's already a suggested action.
3. You are free to decide how and when to end the connection.

Task 2: TCP server

Implement a TCP server program, "server.py" that takes a single parameter *port*. The idea is that the server acts as a chat room and that it initiates rounds of dialogue to make the protocol (yes, you're making a protocol!) easier to implement. You can provide command line parameters if you like, but they must then be explained if you call the program with "--help" or "-h". The server will do the following:

1. Create a TCP socket and bind to localhost, on the port given by *port*
2. Accept any connection. You can expect all connections to be a bot, e.g. that they will not be the first to speak, but that they will always respond. This gives you the option of waiting for them. You can also decide to make your program more robust by reading from clients in parallel, or if you're a real pro, use `select` or `poll` to make the clients non-blocking. But keep in mind: it's better to have something simple that works than something sophisticated that doesn't.
3. Initiate a round of dialogue by suggesting an action. Send the suggestion to each of your connected clients. The action can be random, provided as user input for each.
4. All responses should be sent back out to all clients except the one who sent it.
5. Maintain a list of connected clients. If you want you can let new connections wait until you've completed one round of dialogue. A good program will check if clients are still connected before trying to interact with them. If they're not, or if you decide that they're taking too long to respond, you can remove them from the list of connections.
6. You are free to decide when and how to disconnect the clients (you can even kick them out if they misbehave) and how to gracefully terminate the program.

Suggestions

You are free to add as many details to the program as you'd like, but keep in mind:
Something simple that works is better than something sophisticated that doesn't.
It's often a good idea to make a very simple, fully functional implementation first and then iterate over that and add features and details incrementally. If you run out of time you can then hand in your best working version.

Aim for decent robustness. Nobody expects bug-free code, but you should aim to make the "happy day scenario", e.g. the case when the user behaves well, work without ugly errors.

Some suggestions:

- You can add some delay before your bots respond.
- Randomization is great for making things seem a bit more lifelike.
- You can go crazy with grammar and language patterns and make your bots respond to things in complex ways if you like. They could do anything from telling jokes to trolling, flirting or correcting each other's grammar. Beware of the rabbit hole this may be though!
- You can give your bots memory, and have them remember what others have said in the past.
- You are free to let bots interrupt one another, even mid sentence, and get into arguments with one another. That's not necessarily an easy thing to implement and

get right but your server can also act as moderator, limiting the number of messages each bot can send before making a new suggestion.

- Optionally make a "bot" that takes its response from the command line. That way you or other users can interact with the bots and make the dialogue more interesting.
- Keep output from the program concise. It's often useful for developers to print out a lot of details - that's allowed, but it's better to only print debug information if the user provides a "--verbose" flag or otherwise indicates that they want to see debug output.
    - You could add "private thoughts" to each bot and have them print that to their terminal window without saying it over the connection if you like. That does not count as debug information, although it can also help the developer understand the internal state of the bot.
- Don't nag your users. It's sometimes nice to let the user add choices and options, but your defaults should work well without user interaction. Don't make them fill out forms.
- You are free to change and extend the bot funcion signature or get rid of it all together if you think it makes the program better. For example you could:
    - Choose an object oriented approach and have each bot be implemented as a subclass that inherits an abstract base class / super class or an interface.
    - You could keep bots as functions but pass around a state-keeping object instead, where the message history of each of the other participants could be stored
    - You could have a mix of some primitive bots as described in the introduction and some more advanced bots implemented in a different way.
    - Note: It's tempting to get fancy and clever and use all the techniques you know. You're free to do that, but remember that the primary objective is to demonstrate your mastery of socket programming. Some times less is more, some times more is more.