


 <http://ics32.markbaldw.in/assignments/a2.html>

 13 min read

Chatting with Friends — ICS 32 Winter 2021

Introduction¶

The primary learning outcome for a1 was to provide you with an opportunity to learn how to work with modules from the Python Standard Library, explore recursive functions, and gain familiarity with error handling.

For assignment 2, you are going to build upon your code from the a1 program, by creating a local-first journaling program. This means that your journal entries will be stored locally (on your computer) with an option to be shared to the Internet. Where on the Internet? Any server that supports the ICS32 distributed social platform (DSP) (for now that's just me)!

Summary of Program Requirements¶

1. Manage user data using a file store
2. Design a command line user interface
3. Write code to support a communication protocol
4. Write code to communicate with a remote server over network sockets

Learning Goals¶

1. Working with modules

2. Integrate existing code into your program
3. Communicating with networks and sockets
4. Working with protocols

Program Requirements¶

Unlike the previous assignment, you will have a lot more flexibility in how you design your program's user interface. I will not be providing a validity checker for this assignment, so the input and output of your program is largely up to you. However, there are some conditions:

- You must divide your program into **at least three modules**, not including the Profile module. Your modules should be named and loosely modeled after the descriptions below.
 1. `a2.py`: Your first module will be the entry point to your program.
 2. `ds_protocol.py`: Your adaptation of the DS Protocol.
 3. `ds_client.py`: Your distributed social client module should contain all code required to exchange messages with the DS Server.
- You must *retain* the functionality specified for a1. A user should be able to use your program to locate, create, and load `.dsu` files. If you were unable to get all a1 features to be fully functional, don't worry, you won't be penalized again. Just carry through to a2 whatever you have completed.
- You must also retain the input command format for any new commands your program may need, which if you recall looks like this:

```
[COMMAND] [INPUT] [[-]OPTION] [INPUT]
```



- You must use the `Profile.py` module that accompanies this assignment without modification (more on that below).

Otherwise, you are free to design your program any way you like. This means if you would like to provide more helpful feedback after error conditions, for example, you are free to do so.

The program you will be creating is divided into two parts. I strongly encourage you to read through both parts so that you have a clear picture of what the complete program will do. Then focus on completing part 1 first. Ensure that part 1 is reliably working before continuing on to part 2.

Profile module update

If you struggled with input processing in a1 or would like to compare your approach to ours, we are releasing a demo program that includes two way to process input.

DS Input Processor

You are free to use this code in all remaining assignments.

Part 1¶

Your program should extend the create command that you built in a1. When a user enters the create command, not only will your program create a new DSU file, but it will also use inputs to collect additional data from the user. The collected data should be contained in a Profile object until the user ends the program, at which time the data should be saved to the user created DSU file.

Program Feature

Extend the 'C' create DSU file command to collect profile information about the user and store the profile information using the Profile module.

Profile module update

Bug fixes made on 1/28 at 3:03 PM PST to v0.1.6

The DSU Profile Module

You must use the Profile module (linked above) to store the data you collect from the user. The Profile module has two primary functions: 1) Provides properties to store user input, and 2) Provides encode and decode functions to support saving profile data to a DSU file. To understand how to use Profile module, you will be required to study the code that it contains. Your user interface should collect all of the data represented by the properties in the Profile module.

Code Reading Tip

Asking questions and working together is a critical component to a successful development team. So rather than tell you exactly how to use the Profile class, I want you to help each other figure it out. Although you are **not allowed** to share the code you write for this assignment with each other, you are allowed to share what you learn about the Profile module. My only requirement is that you do so publicly using the a2-qa channel in Zulip. We will keep an eye on conversation and make sure you are all headed in the right direction.

Your program should allow a user to search for and load a DSU file using the input command conventions established in a1. While you should have the search functionality already in place, you will need to add support for loading a DSU file.

Program Feature

Add new command to user interface to support loading DSU files.

As you might recall, in a1 you performed a very similar operation when you implemented the 'R' command. We don't want to change the behavior of the 'R' command, as there may be conditions in which a user might like to check the contents of a DSU file before loading it. Therefore, you will need to create a new input command to support loading an existing DSU file. There are two ways to approach this task: 1) create a new input command (e.g., 'O [path].dsu') or 2) add a load option to the 'R' command (e.g., 'R [path].dsu -l'). The choice is yours, but be sure to inform the user about how they should go about loading a DSU file!

Finally, as this program will be growing in complexity for the rest of the quarter, you should start getting used to writing tests. A good place to start is to ensure that your program is properly loading and saving profile data after it has been supplied by the user.

Part 2¶

Now that you have a program that can successfully store and retrieve user input, it's time to do something with it! For the second part of this assignment, you will be connecting to a server and sending the user data you have collected in Part 1 to the world wide web. You will do this by using sockets.

Much of the work you will need to do to connect to the DS Server using network sockets is covered in the Networks and Sockets lecture. If you haven't watched the lecture or looked through the notes yet, now is a good time!

Unlike the `Profile.py` module that is supplied to you, you will need to create the `ds_client.py` module yourself. However, it must contain and make use of the following function signature.

```
# ds_client.py

def send(server:str, port:int, username:str, password:str, message:str, bio:str=None):
    """
    The send function joins a ds server and sends a message, bio, or both

    :param server: The ip address for the ICS 32 DS server.
    :param port: The port where the ICS 32 DS server is accepting connections.
    :param username: The user name to be assigned to the message.
    :param password: The password associated with the username.
    :param message: The message to be sent to the server.
    :param bio: Optional, a bio for the user.
    """
    pass
```



The validity checking tool that we will use for grading will import your `ds_client.py` module and call this function with randomly generated values. When called, the information we

supply should either successfully transmit the data to the DS Server or, in the case of incorrect data, gracefully inform our program what went wrong.

Program Feature

Connect to, send, and receive information with a remote DSP server

When communicating with a server using sockets, it is common to establish a protocol for exchanging messages. A protocol is a predefined system of rules to transfer information between two or more systems. For a2, your program will need to support the DSP protocol to successfully communicate with the DSP server. Your protocol code should be placed in the `ds_protocol.py` module.

The DSP protocol supports the following send commands:

join

Accepts either an existing user and password, or a new user and password

post

Accepts a journal post for the currently connected user

bio

Accepts a user bio, either adding or replacing existing user bio

All protocol messages must be sent in JavaScript Object Notation (JSON) format. All responses from the DSP server will be in JSON format as well.

Tip

You don't have to concern your self too much with JSON for this class. However, if you would like to learn more about JSON and why it is a great format for storing and transporting data, visit:

JSON.org

The following code snippets demonstrate how each of the DSP commands should be wrapped in JSON. You are free to adopt these templates for your program.

```
# join as existing or new user
{"join": {"username": "ohhimark", "password": "password123", "token": "user_token"}}

# timestamp is generate automatically in the Profile module using Python's
# time.time() function
{"token": "user_token", "post": {"entry": "Hello World!", "timestamp": "1603167689.3928561"}}

# for bio, you will have to generate the timestamp yourself or leave it empty.
{"token": "user_token", "bio": {"entry": "Hello World!", "timestamp": "1603167689.3928561"}}
```



The DSP protocol also supports response commands:

error

Will be received when a send command is unable to be completed

ok

Will be received when a send command is successful

Your program should expect one of these two responses after sending a command. The following code snippet demonstrates how the response command should be wrapped in JSON. Again, you are free to adopt this template for your program.

```
# Error messages will primarily be received when a user has not been
# established for the active session. For example, sending 'bio' or 'post'
# before 'join'
{"response": {"type": "error", "message": "An error message will be contained here."}}

# Ok messages will be received after every successful send command.
# They likely will not be accompanied by a message.
{"response": {"type": "ok", "message": "", "token": "user_token"}}
```



Program Feature

Adapt the DSP protocol for use in your program.

A typical exchange between a program and a DS server might look like the following:

```
join_msg = '{"join": {"username": "ohhimark", "password": "password123", "token": ""}}'

send = client.makefile('w')
recv = client.makefile('r')

send.write(join_msg + '\r\n')
send.flush()

resp = recv.readline()
print(resp)

>>> b{"response": {"type": "ok", "message": "Welcome back, ohhimark", "token": "07da3ddc-6b9a"}}
```



In your Python code, you will treat JSON messages as type string. In the snippets above, you will likely need to replace the hard coded values (e.g., ohhimark, password123, etc.) with the variables in your program that store the actual data you intend to send to the DSP server. There are many ways to do this, but you should focus your efforts on using the string formatting functions found in the Python Standard Library.

DS Server Protocol Helper Code

The following code was added on 2/1 to help you process json formatted messages sent by the server. NOTE: You will need to adapt the example code to work with protocol messages.

To process the messages from the DS server, you may adapt the following code to reduce some of the extra work of parsing strings:


```

import json
from collections import namedtuple

# Create a namedtuple to hold the values we expect to retrieve from json messages.
DataTuple = namedtuple('DataTuple', ['foo', 'baz'])

def extract_json(json_msg:str) -> DataTuple:
    """
    Call the json.loads function on a json string and convert it to a DataTuple object
    """
    try:
        json_obj = json.loads(json_msg)
        foo = json_obj['foo']
        baz = json_obj['bar']['baz']
    except json.JSONDecodeError:
        print("Json cannot be decoded.")

    return DataTuple(foo, baz)

# Example Test
json_msg = '{"foo": "value1", "bar": {"baz": "value2"}}'
print(extract_json(json_msg))

>>> DataTuple(foo='value1', baz='value2')

```



Finally, if you have made it this far, you are probably wondering about the information required to connect to the DSP server. I will have the server up and running very soon. In the meantime, you should be able to test using the echo server covered in lecture. Since this project page is public and I would like to avoid undesirable traffic aimed at my server, I will be communicating server details via Zulip only.

You may also be wondering, what happens to all the messages you send...Well, since we are using networked sockets, you may have guessed, all of your messages will be published to a publicly available web page. So be kind and respectful in the messages you write!

How we will grade your submission¶

This assignment will be graded on a 12-point scale, with the 12 points being allocated completely to whether or not you submitted something that meets all of the above requirements. The following rubric will be used:

Requirements and Function | 8 pts

Does the program do what it is supposed to do?

Does the `ds_client` module function independently of the rest of the program?

Are there any bugs or errors?

Module Usage | 1 pts

Are all required modules named and used as specified?

Quality and Design | 3 pts

Is the code well designed?

Is the code clearly documented?

Now that you have successfully completed on large program, we will start to look at the quality and design criteria a little more closely. If you have not been putting time into organizing and documenting your code, now is a good time to start.

Generated with Reader Mode