What is REPL?

- REPL is an online coding platform that developers can use to run simulated applications / tests without having to worry about installing dependencies on their local machines

- There are different REPL environments which cater to all sorts of simulations developers might want to use e.g. javascript environments, python environments, react + typescript environments, etc.

- Because these environments are limited in what languages they run, it makes them light-weight and easy to scale

Before you do anything in REPL

- Please make sure to fork the original REPL you clicked on in the site. Click on the `fork` button to do this
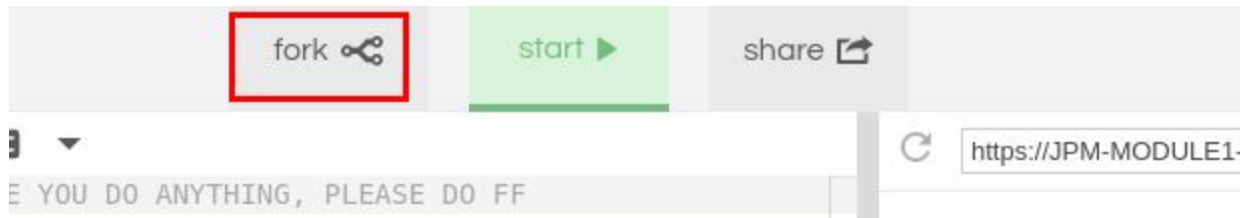
- **Please do not skip the any of the next slides.** They are crucial before you make any changes. Otherwise you might miss the prestep that will allow you create valid patch file!

- **Please avoid refreshing the REPL session** if you do not want to redo changes
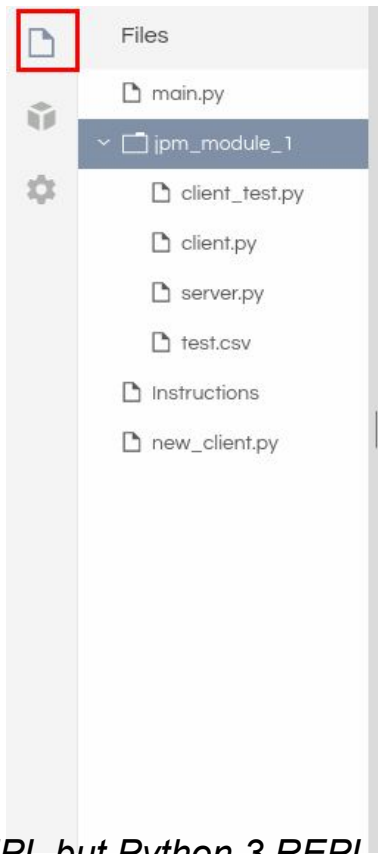
![InsideSherpa]

Basic REPL Navigation

- The left side panel / column contains icons which you can click to help you configure your current environment more. These icons are Files, Packages, Debugger (*only for Python 3 REPL*) and Settings. YOU ONLY NEED TO CARE ABOUT THE `**Files**` icon since clicking this will show you the files that you will use for this exercise
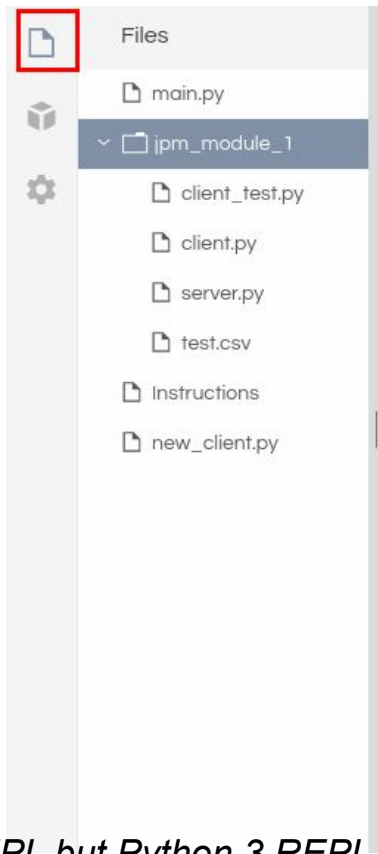
note: the image here is of the Python 2.7 REPL but Python 3 REPL should look similar, just with more files)

Basic REPL Navigation

- When you're able to click the Files icon, it should expand and show you all the files and folders that are included in this environment. These files are the dependencies for you to run the application you'll be working with and modifying to accomplish the task

Files
- main.py
- jpm_module_1
  - client_test.py
  - client.py
  - server.py
  - test.csv
- Instructions
- new_client.py

note: the image here is of the Python 2.7 REPL but Python 3 REPL should look similar, just with more files)

InsideSherpa

Basic REPL Navigation

- You can click any of the files and folders to see their contents. DO NOT MAKE ANY CHANGES YET (guidelines below will help you do that step by step). Try clicking on some files. Don't worry, you can click back to the instructions file.

Files

- main.py
- jpm_module_1
    - client_test.py
    - client.py
    - server.py
    - test.csv
- Instructions
- new_client.py

note: the image here is of the Python 2.7 REPL but Python 3 REPL should look similar, just with more files)

InsideSherpa

Basic REPL Navigation

- Contents of the files you access will be shown in the center column / area of the screen. This area is also where you can make edits to the contents / code you will see that's inside the files you access.

note: the image here is of the Python 2.7 REPL but Python 3 REPL should look similar, just with more files)

Basic REPL Navigation

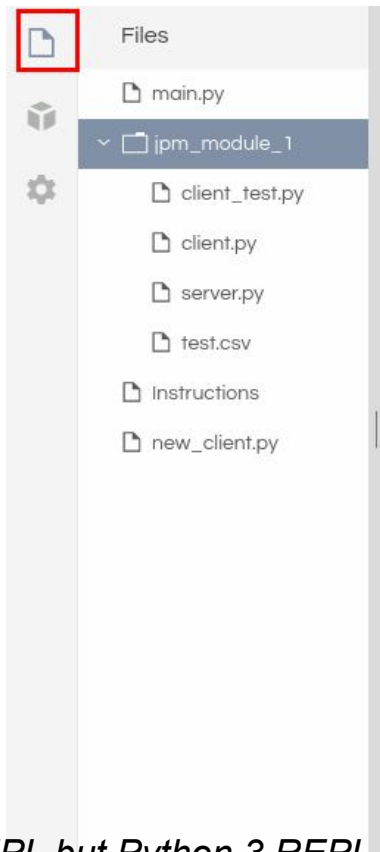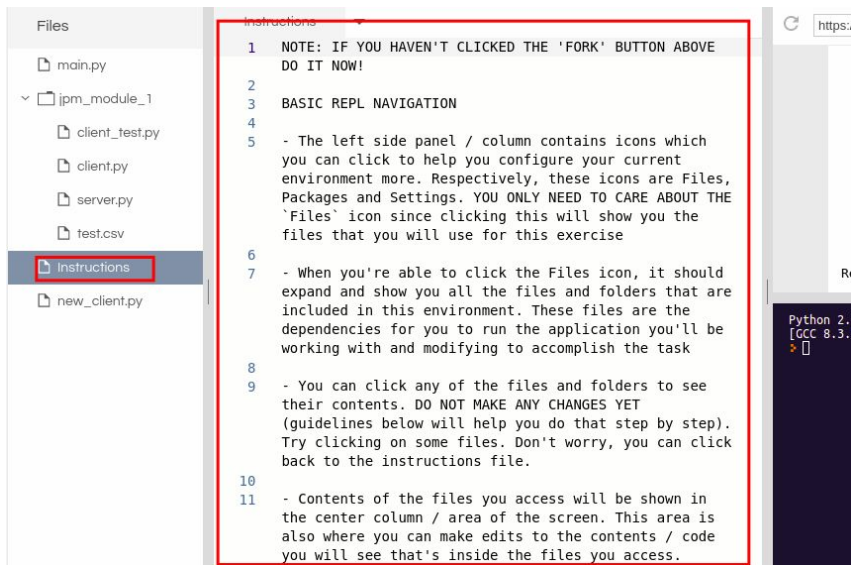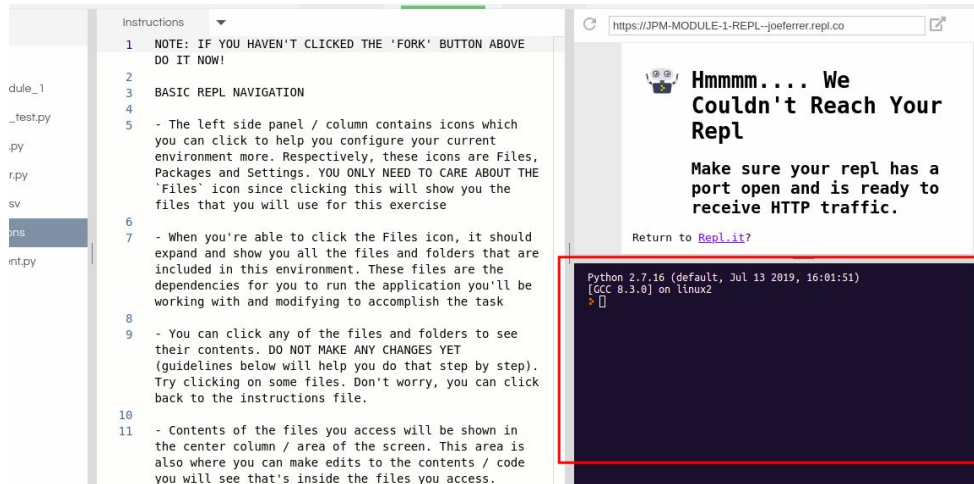- On the right most area, i.e. the blue colored screen, is the command line. It's mainly the area where the output of the application will be shown but you can also type commands in there. Since the environment is in Python, you can type Python code/commands there too to test out

note: the image here is of the Python 2.7 REPL but Python 3 REPL should look similar, just with more files)

Basic REPL Navigation

- The 'Start' button will always just run what's in main.py. You can't run other scripts with the 'Start' button.

- The 'Share' button will allow you to share your REPL instance with others should you need assistance
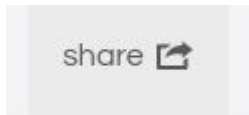
- The browser-like output on the upper right is just for cases if we're hosting a browser like application. For this task we're not. So don't mind it
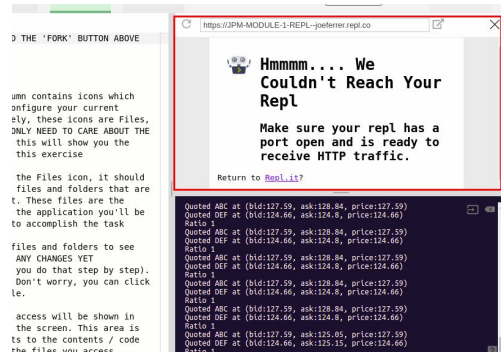
Basic REPL Navigation

- Congrats you now know how to navigate REPL! Please move to the next set of slides that will discuss Basic REPL editing

Basic REPL Editing

- Code/text editing in REPL is pretty straightforward. You just have to click on middle section of the screen and start typing

- DO NOTE THAT: We discovered REPL has this quirk when it comes to python indent detection: https://repl.it/talk/ask/Python-spacing-and-tab-issue/9425

- For the issue mentioned above we have TWO viable workarounds

Basic REPL Editing

- **WORKAROUND OPTION 1**: Since you mainly want to modify client.py, you can first manually type in the methods in there into new_client.py to avoid tab/indentation issues. It's just a short file so it's easy (y*ou can disregard copying the comments you don't need*).

  Then to make things work, assuming you've finished the '[initial steps before making any changes](#)' and you've finished making your changes in new_client.py:
  - remove/delete client.py inside the jpm_module_1 folder
  - rename new_client.py to just client.py
  - and move it inside jpm_module_1

Basic REPL Editing

- **WORKAROUND OPTION 2:** When you make changes, remove all previous comments and start from the `def <function_name>:` line and hit enter, then add your changes step by step following the right indentation. It should work properly.Here's a link to a gif/screencast that shows the workaround: **http://tiny.cc/u6j2ez**

Basic REPL Editing

- A few last notes about the python file(s), i.e. files ending in **.py**, you'll be editing. If you're not familiar with Python, that's just fine. We'll walk you through the changes. All you need to know are just few terms we use in this guide:

- **Comments**: Comments in python are preceded with the **#** symbol or **"""**

```
# BEFORE YOU DO ANYTHING, PLEASE DO FF
# CLICK THE 'FORK' BUTTON ABOVE
```

```
""" ------------- Update this function ------------- """
```

To uncomment these, simply remove the preceding symbols. That said, **do not uncomment unless instructed to.**

Basic REPL Editing

- **Methods/Functions**: We use methods and functions interchangeably in this guide. Basically methods / functions are blocks of code that are encapsulated by a **def.** An example of a method is the pic below...

```python
def getRatio(price_a, price_b):
    """ Get ratio of price_a and price_b """
    """ ------------- Update this function ------------- """
    """ Also create some unit tests for this function in
    client_test.py """
    return 1
```

Introduction to task

Hi! Welcome to your first task in the JP Morgan Program hosted by Insidesherpa

For this task you will need to come up with a git patch file which you will have to upload in the first module page of the JP Morgan Program

A git patch file is just a file that we can apply to a git repository to check if your code changes will arrive at the correct output that's needed from you to complete this task.

Introduction to task

The instructions in the next slides will help you through this task. The procedure is divided into a sequence of stages so that you can arrive at the solution in a methodical manner...

How to use this guide

This guide will help you finish the task by walking you through a series of steps. You can keep referring to this guide as you do the exercise by just having it open in a separate tab or browser window.

To access the contents of other files/folders in the REPL, make sure the REPL tab / window is active and it's as simple as clicking on the file/folder you want to look at.

Brief info about the files in this REPL

- **server.py**: aka **server3.py** in the python3 REPL. This is the file that will simulate a server application waiting for requests so that it can send back data about stocks

- **client.py**: aka **client3.py** in the python3 REPL. This is the application that will contact server.py to process info about the stocks data. This is the only file you are required to touch for this task

Brief info about the files in this REPL

- **client_test.py**: This is a unit test script that is independent of server.py and client.py. It just allows developers like you to ensure the methods you've defined / modified are working as expected. This file is part of the bonus task should you wish to do it

- **new_client.py**: aka **new_client3.py** in the Python3 REPL (see BASIC REPL EDITING section to remember what this file is for)

Brief info about the files in this REPL

- **test.csv:** This is where the stocks data that the server returns to your client comes from

- **main.py:** is the script that runs in REPL that runs both server.py and client.py. the output you see in the blue screen comes from client.py. YOU DO NOT NEED TO MAKE ANY CHANGES TO THIS SCRIPT. IF YOU WISH TO DO THE BONUS TASK, THATS THE ONLY TIME YOU MIGHT NEED TO UNCOMMENT STUFF HERE (SEE COMMENTED INSTRUCTIONS IN main.py)

- Other files: You do not need to bother about other files. They are artifacts used by REPL to install dependencies before running the applications

Set Up

This platform is an alternative to the first way of accomplishing module 1.

Using this platform, repl.it, you don't have to do a lot of set up or installation of dependencies. Python, the scripting language that is used for this exercise, is already available and its dependencies / packages will be installed automatically when you run the main.py script via the 'Start' button

Objectives

- This part assumes you haven't made any changes to the code yet.
- Run the application by hitting the 'Start' button. You should end up with an output like:

```
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:125.05, price:127.59)
Quoted DEF at (bid:124.66, ask:125.15, price:124.66)
Ratio 1
>
```

Objectives

- If you closely inspect the output in the previous slide, there are two incorrect things…
  - (1) Ratio is always 1
  - (2) The price of each stock is always the same as its bid_price.

- These are obviously wrong so you job is to fix those things…

- Don't worry we'll walk you through how to get these things done
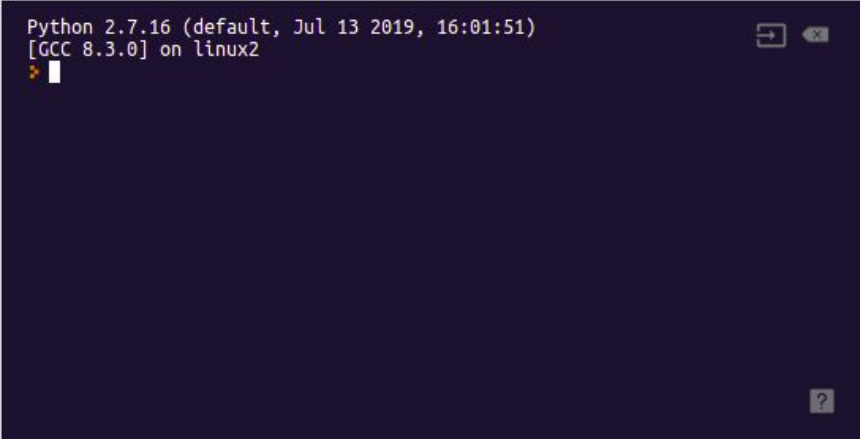
Where to make changes:

- All the changes you have make to get the right output will be in the client.py file inside the jpm_module_1 folder (client3.py if you're in python3 REPL)

- If you wish to do the bonus task after accomplishing the main objective, you need to also add some more code in the client_test.py file (we have a guide on how to do that later on).

Before you start coding

First you have to ensure you start with a clean repository where you'll generate your patch file containing your changes later.

On the terminal at the right side (i.e. blue area). Type the following commands (see next slide)

```
Python 2.7.16 (default, Jul 13 2019, 16:01:51)
[GCC 8.3.0] on linux2
>
```

note: the image here is of the Python 2.7 REPL but Python 3 REPL should look similar

Before you start coding

(one line means one command; hit enter on the blue screen when you finish typing a command)

```
import os
os.system('bash')
cd jpm_module_1
git init
git add -A
git config user.email "<your_email_address>"
git config user.name "<your_name>"
git commit -m 'INIT'
exit
```

note: only use this command if your terminal is not yet in the directory itself. **cd** is for changing directory.You will know you're already in the directory if your terminal looks like

runner@7d8985fcef48:~/jpm_module_1$

note: make sure you use your personal email address and your real name

Before you start coding

```
> import os
import os
> os.system('bash')
runner@7d8985fcef48:~$ cd jpm_module_1/
runner@7d8985fcef48:~/jpm_module_1$ git init
Initialized empty Git repository in /home/runner/jpm_module_1/.git/
runner@7d8985fcef48:~/jpm_module_1$ git add -A
runner@7d8985fcef48:~/jpm_module_1$ git config user.email "joe@insidesher
pa.com"
runner@7d8985fcef48:~/jpm_module_1$ git config user.name "joe ferrer"
runner@7d8985fcef48:~/jpm_module_1$ git commit -m 'INIT'
[master (root-commit) c732383] INIT
 4 files changed, 2229 insertions(+)
 create mode 100644 client.py
 create mode 100644 client_test.py
 create mode 100644 server.py
 create mode 100644 test.csv
runner@7d8985fcef48:~/jpm_module_1$ exit
exit
0
>
```

Before you start coding

Avoid refreshing this page from here on. Otherwise you'll have to do these steps again. If you restart your PC or browser, you'll have to do this initial steps again as REPL does not save the entire state of the application you're working with

Making changes in `client.py` (client3.py for python3)

- To get the correct computation of stock price and ratio between prices, you need to make changes in `client.py` inside the jpm_module_1 folder.

- Click that folder and click the file named `client.py`

- Once you see the contents of `client.py`, you can make changes by typing on the screen (but factor in workaround 2) or you can manually copy the code into new_client.py (using workaround 1)

Making changes in `client.py` (client3.py for python3)

- The changes you need to make will be in the following methods of the file
  - getDataPoint
  - getRatio
  - Main

- The changes for each method will be dissected for each method on the next slide

Making changes in `client.py` (client3.py for python3) **getDataPoint**

getDataPoint. In this method, you'll have to make the modifications to compute for the right stock price. This means you have to change how `price` is computed for. The formula is `(bid_price+ask_price) / 2`.

YOU DO NOT NEED TO CHANGE the return value as that is representational of the entire data point. You should end up with something like:

```python
32 def getDataPoint(quote):
33     """ Produce all of the needed values to generate a datapoint """
34     """ ------------ Update this function ------------ """
35     stock = quote['stock']
36     bid_price = float(quote['top_bid']['price'])
37     ask_price = float(quote['top_ask']['price'])
38     price = (bid_price + ask_price)/2
39     return stock, bid_price, ask_price, price
```

Making changes in `client.py` (client3.py for python3) getRatio

getRatio. In the original case, this method just returns 1 all the time. To correct this, you must change the return value to the ratio of stock **price_a** to stock **price_b**

```
41  def getRatio(price_a, price_b):
42          """ Get ratio of price_a and price_b """
43          """ ------------ Update this function ------------ """
44          """ Also create some unit tests for this function in client_test.py """
45          if (price_b == 0):
46                  # when price_b is 0 avoid throwing ZeroDivisionError
47                  return
48          return price_a/price_b
```

note: that we've also added the condition of the case where in price_b could be zero, i.e. division by zero, in the rare chance that it might happen...

Making changes in `client.py` (client3.py for python3) **main**

main method. Now that you've fixed the two other methods, it's just a matter of printing the correct values. For every iteration in the main method, you need to store the datapoints you get from getDataPoint method so that you can properly call getRatio and print the right ratio out: (*the below image is for python 2.7.x version*)

```python
50  # Main
51  if __name__ == "__main__":
52
53          # Query the price once every N seconds.
54          for _ in xrange(N):
55                  quotes = json.loads(urllib2.urlopen(QUERY.format(random.random())).read())
56
57                  """ ---------- Update to get the ratio -------------- """
58                  prices = {}
59                  for quote in quotes:
60                          stock, bid_price, ask_price, price = getDataPoint(quote)
61                          prices[stock] = price
62                          print "Quoted %s at (bid:%s, ask:%s, price:%s)" % (stock, bid_price, ask_price, price)
63
64                  print "Ratio %s" % getRatio(prices['ABC'], prices['DEF'])
```

Making changes in `client.py` (client3.py for python3) **main**

main method. (*the image here is now for python 3.x version*)


```python
50    # Main
51    if __name__ == "__main__":
52
53        # Query the price once every N seconds.
54        for _ in range(N):
55            quotes = json.loads(urllib.request.urlopen(QUERY.format(random.random())).read())
56
57            """ ----------- Update to get the ratio -------------- """
58            prices = {}
59            for quote in quotes:
60                stock, bid_price, ask_price, price = getDataPoint(quote)
61                prices[stock] = price
62                print ("Quoted %s at (bid:%s, ask:%s, price:%s)" % (stock, bid_price, ask_price, price))
63
64            print ("Ratio %s" % (getRatio(prices['ABC'], prices['DEF'])))
65
```

python 2.7.x uses xrange.
in python 3.x we use range

There's also a slight difference in the urllib that's used for python3 but you don't have to bother with this

python 2.7.x print does not enclose the text it will print

python3.x encloses the text it prints in parenthesis

Making changes in `client.py` (client3.py for python3) main

- To review the changes in main (*whether it was in python2 or python3*), what we did was create a **prices** dictionary to store the stock prices. Think of a dictionary as a key-value store wherein you can specify a key and be able to retrieve a value. In our case, they key was the stock name and the value was the price.

- We then used this **prices** dictionary at the end to pass in the right values in the **getRatio** function.

Bonus Task: **client_test.py**

- Click link if you want to do bonus task
  https://insidesherpa.s3.amazonaws.com/vinternships/companyassets/Sj7tem
  L583QAYpHXD/client_test_m1_v1a.pdf

- If you wish to do this, changes will be similar in REPL as it discusses how to
  change the client_test.py. However you must first uncomment the following
  code in the main.py of the REPL you're in (see next slide)

Bonus Task: **client_test.py**

```
27    # FOR BONUS TASK
28    # IF YOU WANT TO DO IT THEN UNCOMMENT THE CODE BELOW
29    # Comments are anything that's preceded with '#'
30    # TO UNCOMMENT JUST REMOVE THE '#'
31
32    #print("UNIT TEST RESULTS BELOW...")
33    #process2 = subprocess.Popen(['python',
      #'client_test.py'], cwd=os.getcwd(),
      #preexec_fn=os.setsid)
34    #process2.wait()
```

To uncomment, just remove the **#**
note: this image is from the main.py file (towards the end)

Bonus Task: **client_test.py**

- Also, do not make the patch file yet if you wish to do the bonus task.
- Avoid restarting your REPL session too! Or you'll have to do the initialization steps again and redo your changes...

Creating your patch file

- Once your confident with your changes, you can now generate your patch file.
- To do that, click on the terminal at the right again and do the following commands on the blue REPL terminal (*one line, one command; hit enter on the blue screen when you finish typing a command*):

```
import os
os.system('bash')
cd jpm_module_1
git add -A
git commit -m 'Create Patch File'
git format-patch -1 HEAD
mv <name of patch created> ../.
exit
```

See the picture on the next slide to understand better

Creating your patch file

```
> import os
> os.system('bash')
runner@5576a83ae8cc:~/jpm_module_1$ git add -A
runner@5576a83ae8cc:~/jpm_module_1$ git commit -m "Create Patch File"
[master 4f1517f] Create Patch File
 1 file changed, 1 insertion(+), 1 deletion(-)
runner@5576a83ae8cc:~/jpm_module_1$ git format-patch -1 HEAD
0001-Create-Patch-File.patch
runner@5576a83ae8cc:~/jpm_module_1$ mv 0001-Create-Patch-File.patch ../.
runner@5576a83ae8cc:~/jpm_module_1$
```

- Notice we didn't have to execute **cd jpm_module_1** anymore because our bash opened inside the **jpm_module1 directory already.**
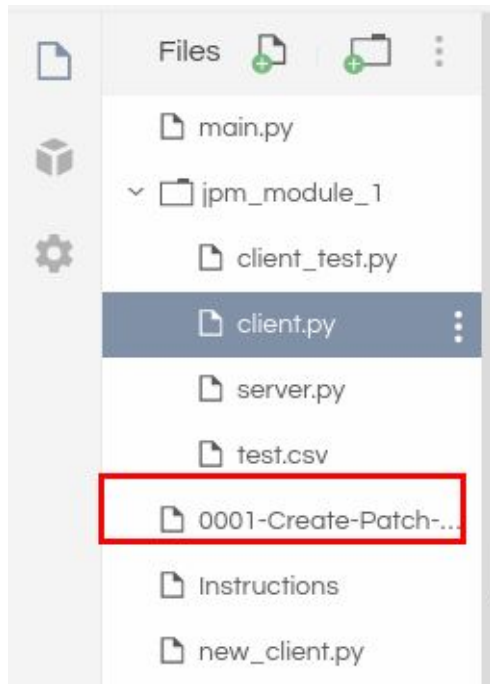
```
> os.system('bash')
runner@5576a83ae8cc:~/jpm_module_1$
```

- When you run the **mv 001-Create-Patch-File.patch ../.** you should see the patch file in your files sidebar (see image next slide)

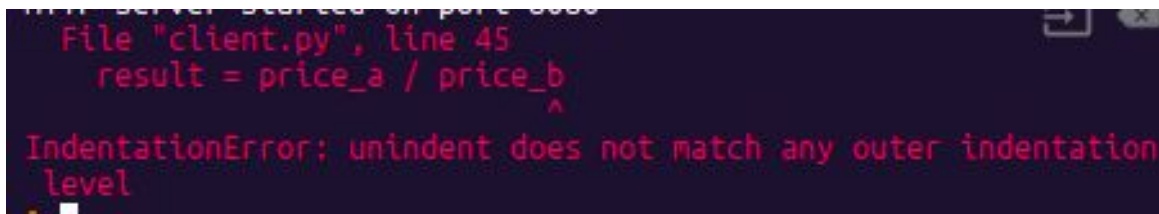# Creating your patch file

Creating your patch file

- Download the files by clicking on the 3 vertical dots beside the new folder button on the upper left side of the screen.
- This will download all the files to your PC as a **.zip** file
- Since we only want the **.patch** file, all you have to do is unzip the **.zip** file on your local machine and pick the **.patch** file there
- Finally, **just submit the .patch file on the JPM module 1 page.**

- **And you're done!**
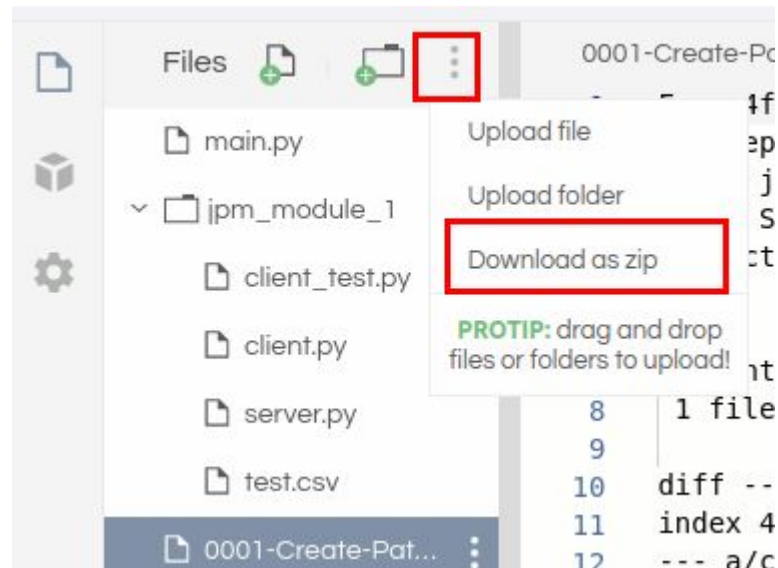
Troubleshooting



```
    File "client.py", line 45
        result = price_a / price_b
                                  ^
IndentationError: unindent does not match any outer indentation
  level
```

If you encountered a similar error like the image above, please re-read the Basic REPL Editing section, particularly about the workarounds as the possible solutions are discussed there...

Support

- If you encountered any issues, please post your issue/inquiry in any of the ff places:
  - https://github.com/insidesherpa/JPMC-tech-task-1/issues
  - The support button in the JPM Module 1 page
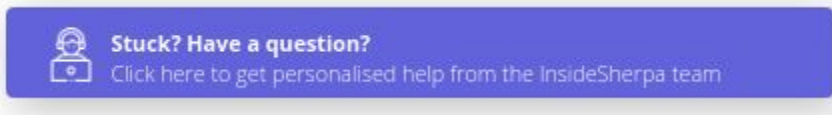
- When submitting a query, please don't forget to provide as much context as possible, i.e. your OS, your browser, what you've done here in REPL, what your errors is/are, etc. (screenshots would help too)