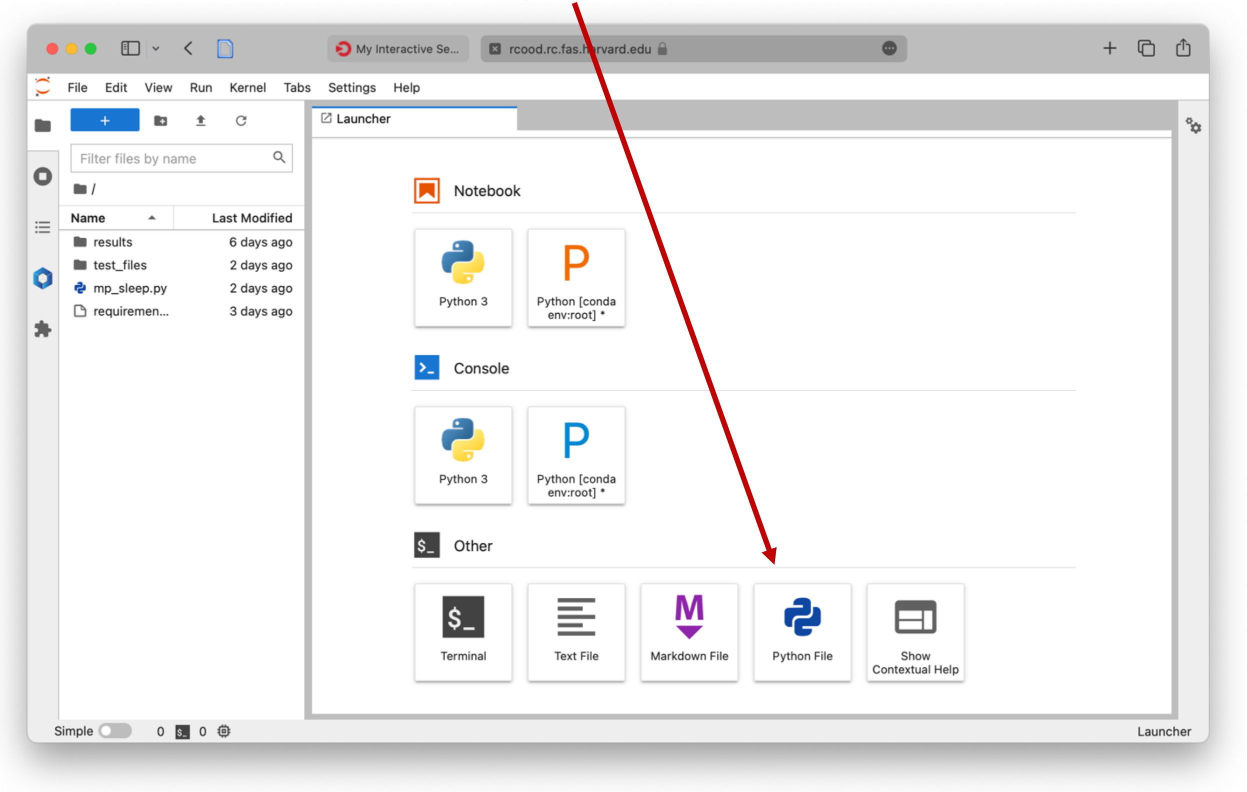
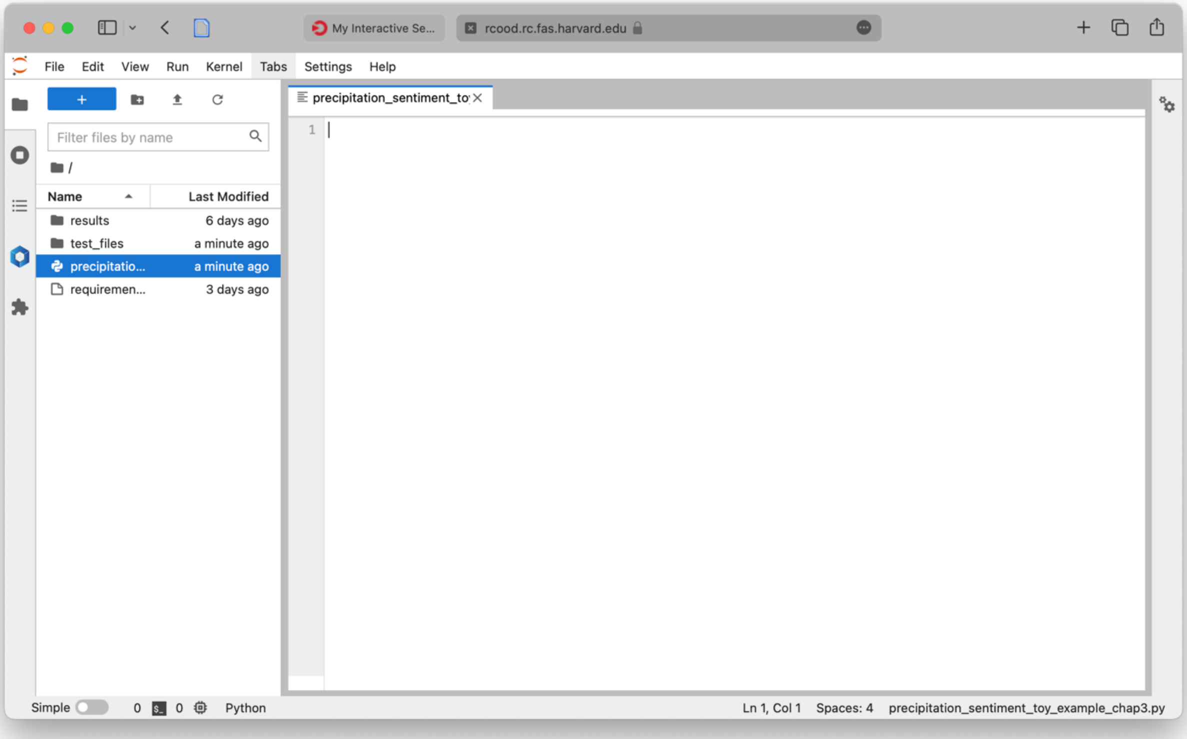
Exercise 3

1. Run your Python program on a compute-node

Start a new Jupyter session via the FASRC web interface, see Exercise 1 for help. In the “Launcher”, in section “Other”, select “Python file”:

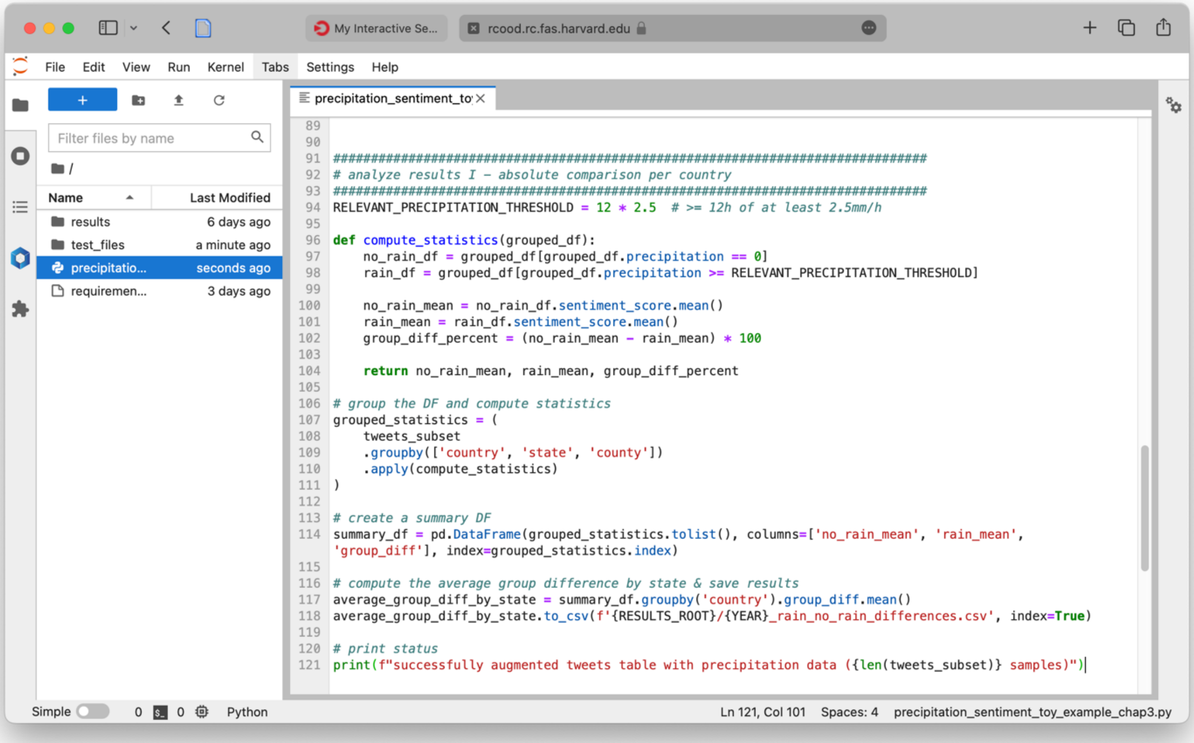


Rename the new file and copy your modified code into the window:



Name	Last Modified
results	6 days ago
test_files	a minute ago
precipitatio...	a minute ago
requiremen...	3 days ago

Ln 1, Col 1 Spaces: 4 precipitation_sentiment_toy_example_chap3.py



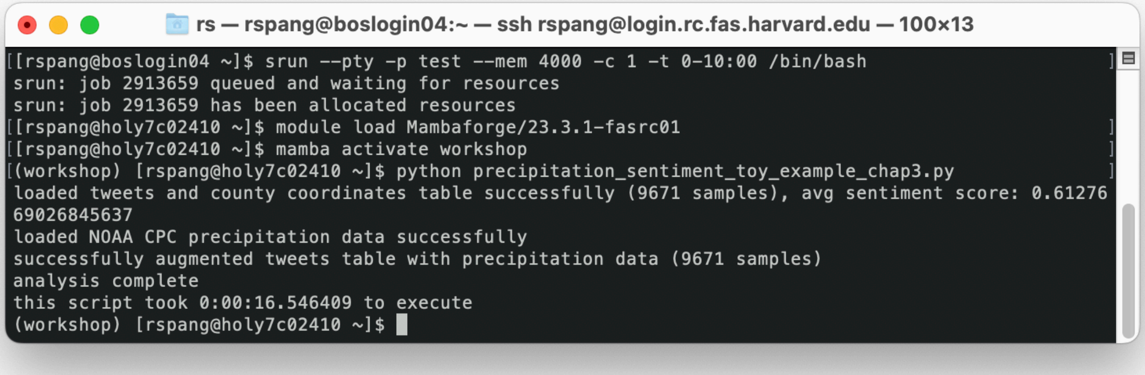
Name	Last Modified
results	6 days ago
test_files	a minute ago
precipitatio...	seconds ago
requiremen...	3 days ago

```python


################################################################################
# analyze results I - absolute comparison per country
################################################################################
RELEVANT_PRECIPITATION_THRESHOLD = 12 * 2.5  # >= 12h of at least 2.5mm/h

def compute_statistics(grouped_df):
    no_rain_df = grouped_df[grouped_df.precipitation == 0]
    rain_df = grouped_df[grouped_df.precipitation >= RELEVANT_PRECIPITATION_THRESHOLD]

    no_rain_mean = no_rain_df.sentiment_score.mean()
    rain_mean = rain_df.sentiment_score.mean()
    group_diff_percent = (no_rain_mean - rain_mean) * 100

    return no_rain_mean, rain_mean, group_diff_percent

# group the DF and compute statistics
grouped_statistics = (
    tweets_subset
    .groupby(['country', 'state', 'county'])
    .apply(compute_statistics)
)

# create a summary DF
summary_df = pd.DataFrame(grouped_statistics.tolist(), columns=['no_rain_mean', 'rain_mean',
'group_diff'], index=grouped_statistics.index)

# compute the average group difference by state & save results
average_group_diff_by_state = summary_df.groupby('country').group_diff.mean()
average_group_diff_by_state.to_csv(f'{RESULTS_ROOT}/{YEAR}_rain_no_rain_differences.csv', index=True)

# print status
print(f"successfully augmented tweets table with precipitation data ({len(tweets_subset)} samples)")
```

Ln 121, Col 101 Spaces: 4 precipitation_sentiment_toy_example_chap3.py

This is a convenient way to update your code quickly. Make sure you update the paths to the datasets: `/n/holyscratch01/cga/rspang/workshop_data/`

To run the code, use a SSH connection to a login-node and request a new interactive session. Since we now work with real data, make sure to allocate enough memory. For the example, 4GB are sufficient. `srun --pty -p test --mem 4000 -c 1 -t 0-10:00 /bin/bash`


rs — rspang@boslogin04:~ — ssh rspang@login.rc.fas.harvard.edu — 100×13

```
[[rspang@boslogin04 ~]$ srun --pty -p test --mem 4000 -c 1 -t 0-10:00 /bin/bash
 srun: job 2913659 queued and waiting for resources
 srun: job 2913659 has been allocated resources
[[rspang@holy7c02410 ~]$ module load Mambaforge/23.3.1-fasrc01
[[rspang@holy7c02410 ~]$ mamba activate workshop
[(workshop) [rspang@holy7c02410 ~]$ python precipitation_sentiment_toy_example_chap3.py
 loaded tweets and county coordinates table successfully (9671 samples), avg sentiment score: 0.61276
 69026845637
 loaded NOAA CPC precipitation data successfully
 successfully augmented tweets table with precipitation data (9671 samples)
 analysis complete
 this script took 0:00:16.546409 to execute
 (workshop) [rspang@holy7c02410 ~]$ 
```

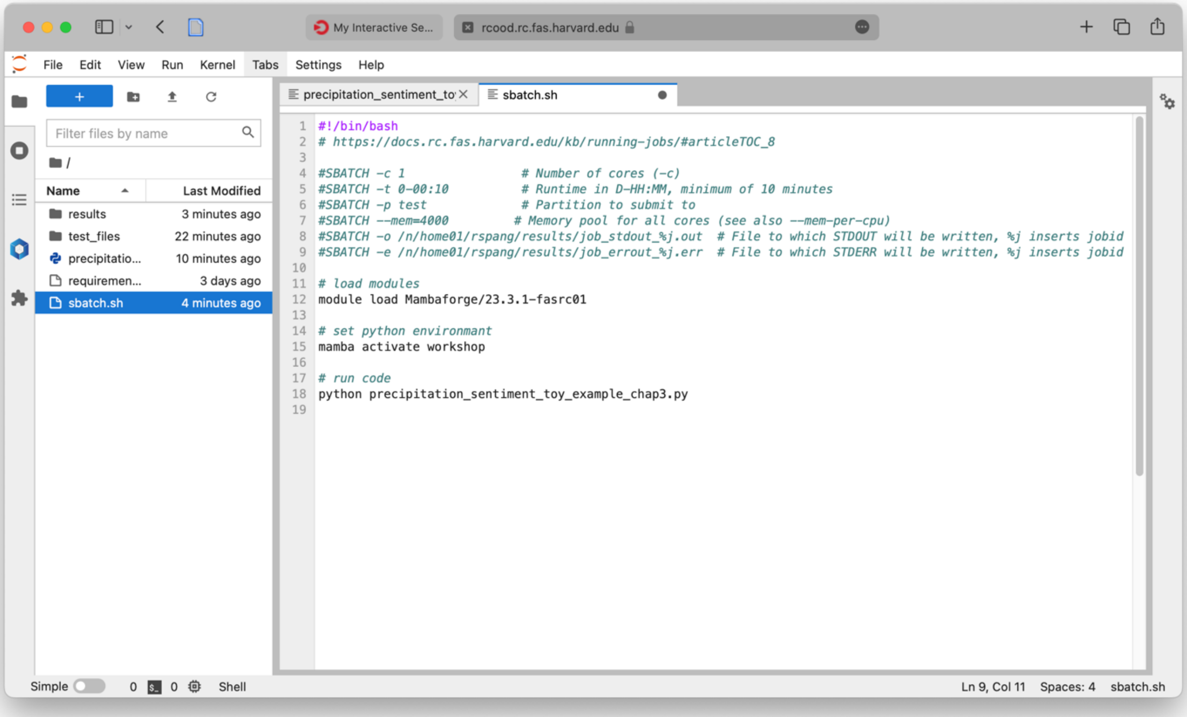
1.2 Add time tracking

To monitor how long the execution of a segment in your code takes, you can take the time before the segment in question (or the entire code), and after. Then, you subtract the first timestamp from the second one to compute the difference. For example:

```python
from datetime import datetime

# take start time
start_execution_timestamp = datetime.now()

a_long_running_task() # replace this with your actual code

# take end time
end_execution_timestamp = datetime.now()
time_diff = end_execution_timestamp - start_execution_timestamp

# print status
print(f"this script took {time_diff}sec to execute")
```

2. Run your Python program on a compute-node as a job

Create a sbatch script to run your Python program. This sbatch script can then be submitted as a SLURM job. The GitHub repo provides you with a template, the cheat sheet also explains the structure of a sbatch script.

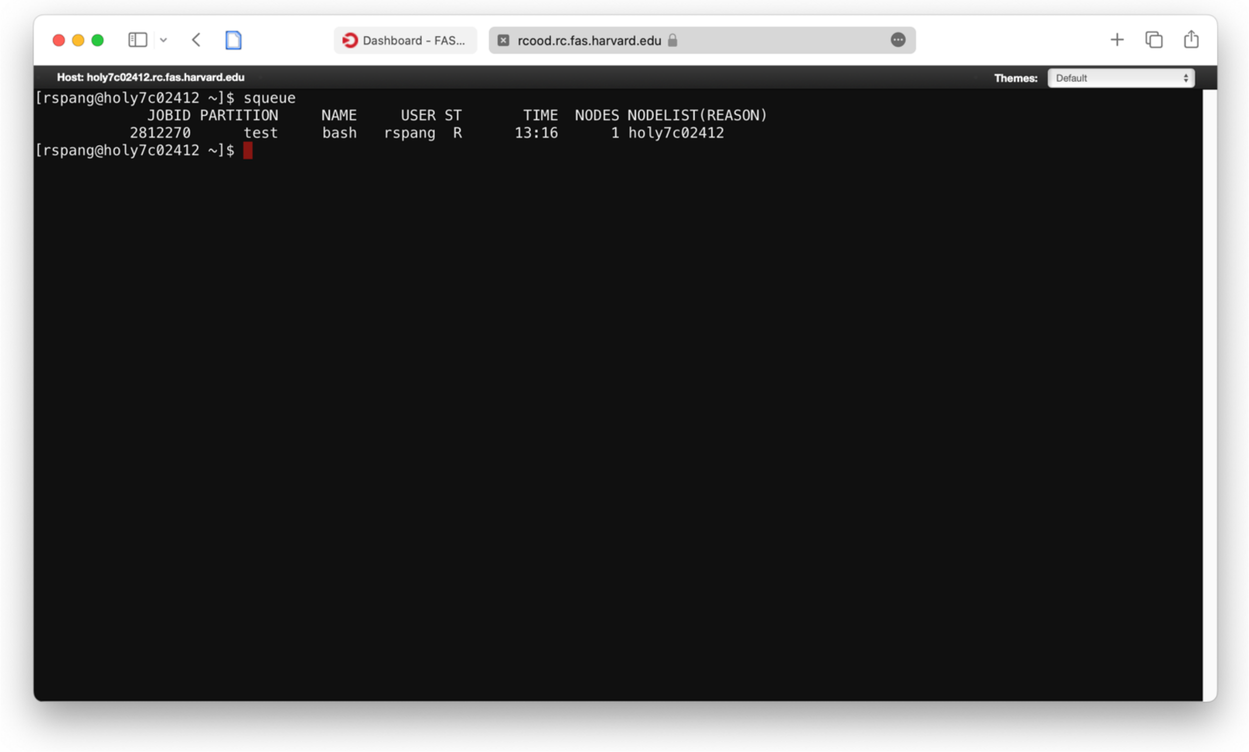


```
#!/bin/bash
# https://docs.rc.fas.harvard.edu/kb/running-jobs/#articleTOC_8

#SBATCH -c 1                # Number of cores (-c)
#SBATCH -t 0-00:10          # Runtime in D-HH:MM, minimum of 10 minutes
#SBATCH -p test             # Partition to submit to
#SBATCH --mem=4000         # Memory pool for all cores (see also --mem-per-cpu)
#SBATCH -o /n/home01/rspang/results/job_stdout_%j.out  # File to which STDOUT will be written, %j inserts jobid
#SBATCH -e /n/home01/rspang/results/job_errout_%j.err  # File to which STDERR will be written, %j inserts jobid

# load modules
module load Mambaforge/23.3.1-fasrc01

# set python environmant
mamba activate workshop

# run code
python precipitation_sentiment_toy_example_chap3.py

```

Once created, the job can be submitted (on a login-node): `sbatch sbatch.sh`

3. Monitor jobs

The SLURM command `squeue` returns a list of all currently running jobs. This will return (at least) the one interactive session you used to start the python program.



Host: holy7c02412.rc.fas.harvard.edu

Themes: Default

```
[rspang@holy7c02412 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           2812270      test     bash   rspang  R      13:16      1 holy7c02412
[rspang@holy7c02412 ~]$ 
```

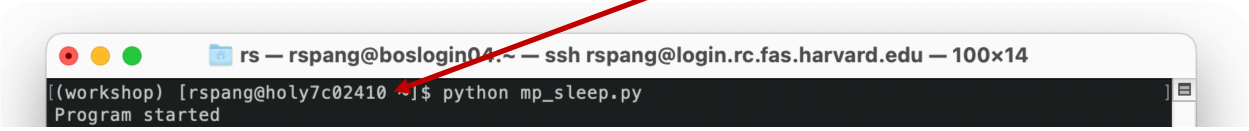
3.1 Monitor the execution of interactive shells

An interactive shell connection runs in your terminal; while a program is running, you cannot use the same terminal for different purposes. However, you can start a second terminal and start a second SSH connection. While this will allow you to run SLURM monitoring tools, it is not guaranteed that you will be connected to the same node if you request a second (interactive) session with SLURM.

However, there is a web-tool that allows you to connect to the same instance from your browser! Open the following link

```
https://rcood.rc.fas.harvard.edu/pun/sys/shell/ssh/COMPUTE_NODE.rc.fas.harvard.edu
```

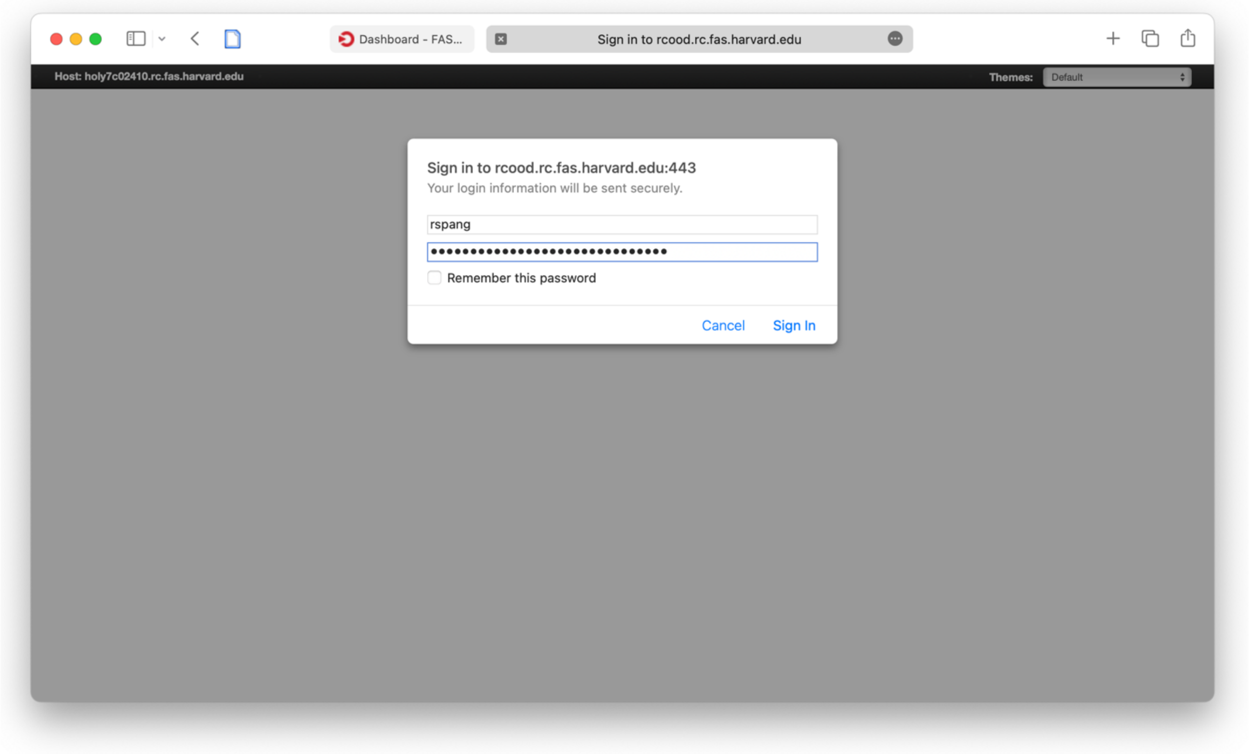
in a browser; replace “COMPUTE_NODE” with the node-ID you are connected to.


rs — rspang@boslogin04:~ — ssh rspang@login.rc.fas.harvard.edu — 100×14

```
[(workshop) [rspang@holy7c02410 ~]$ python mp_sleep.py
 Program started
```

In this example, the node-ID is be “`holy7c02410`”.

If you are asked to login first, provide your FASRC credentials:



Dashboard - FAS...
Sign in to rcood.rc.fas.harvard.edu
Host: holy7c02410.rc.fas.harvard.edu
Themes: Default

Sign in to rcood.rc.fas.harvard.edu:443

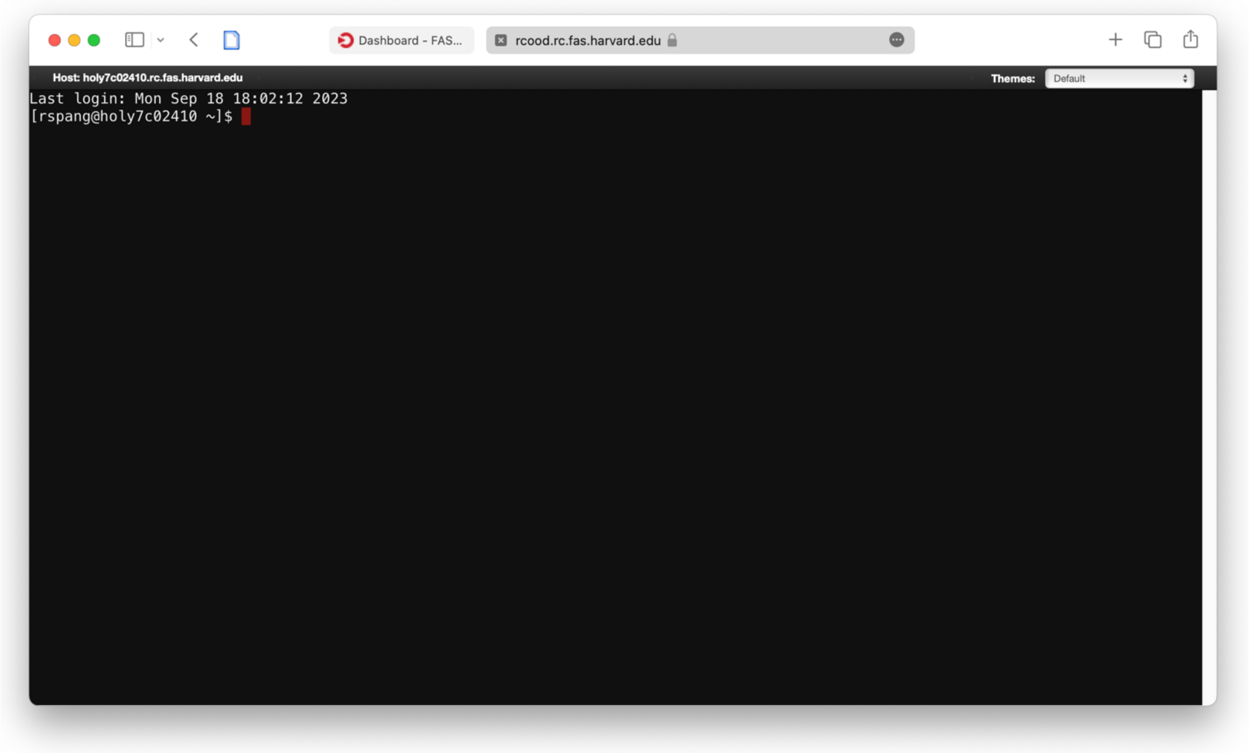
Your login information will be sent securely.

rspang

••••••••••••••••••••••••••••••

□ Remember this password

Then, a terminal-like web page shows up.

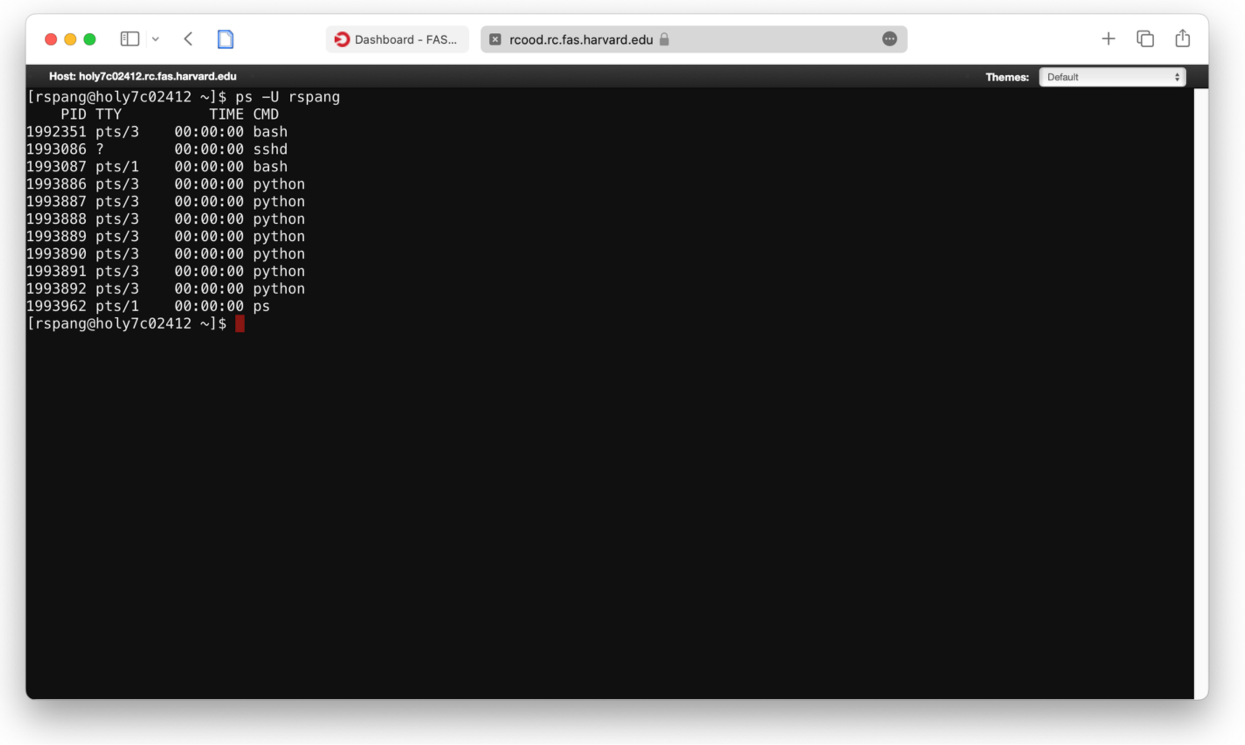


```
Last login: Mon Sep 18 18:02:12 2023
[rspang@holy7c02410 ~]$ 
```

Here, you can run all commands as you would in a normal terminal. For example, you can monitor how many processes you are currently running. Use the following command to obtain a list of all processes in your name:

```
ps -U username
```

Replace “`username`” with your FASRC username.



Host: holy7c02412.rc.fas.harvard.edu

```
[rspang@holy7c02412 ~]$ ps -U rspang
    PID TTY          TIME CMD
1992351 pts/3    00:00:00 bash
1993086 ?        00:00:00 sshd
1993087 pts/1    00:00:00 bash
1993886 pts/3    00:00:00 python
1993887 pts/3    00:00:00 python
1993888 pts/3    00:00:00 python
1993889 pts/3    00:00:00 python
1993890 pts/3    00:00:00 python
1993891 pts/3    00:00:00 python
1993892 pts/3    00:00:00 python
1993962 pts/1    00:00:00 ps
[rspang@holy7c02412 ~]$
```

What you learned in this exercise:

- How to create a python script using the Jupyter web interface
- How to create a sbatch script and how to submit a SLURM job
- How to start a browser-based shell to connect to an ongoing (interactive) session
- How to monitor processes and jobs while running