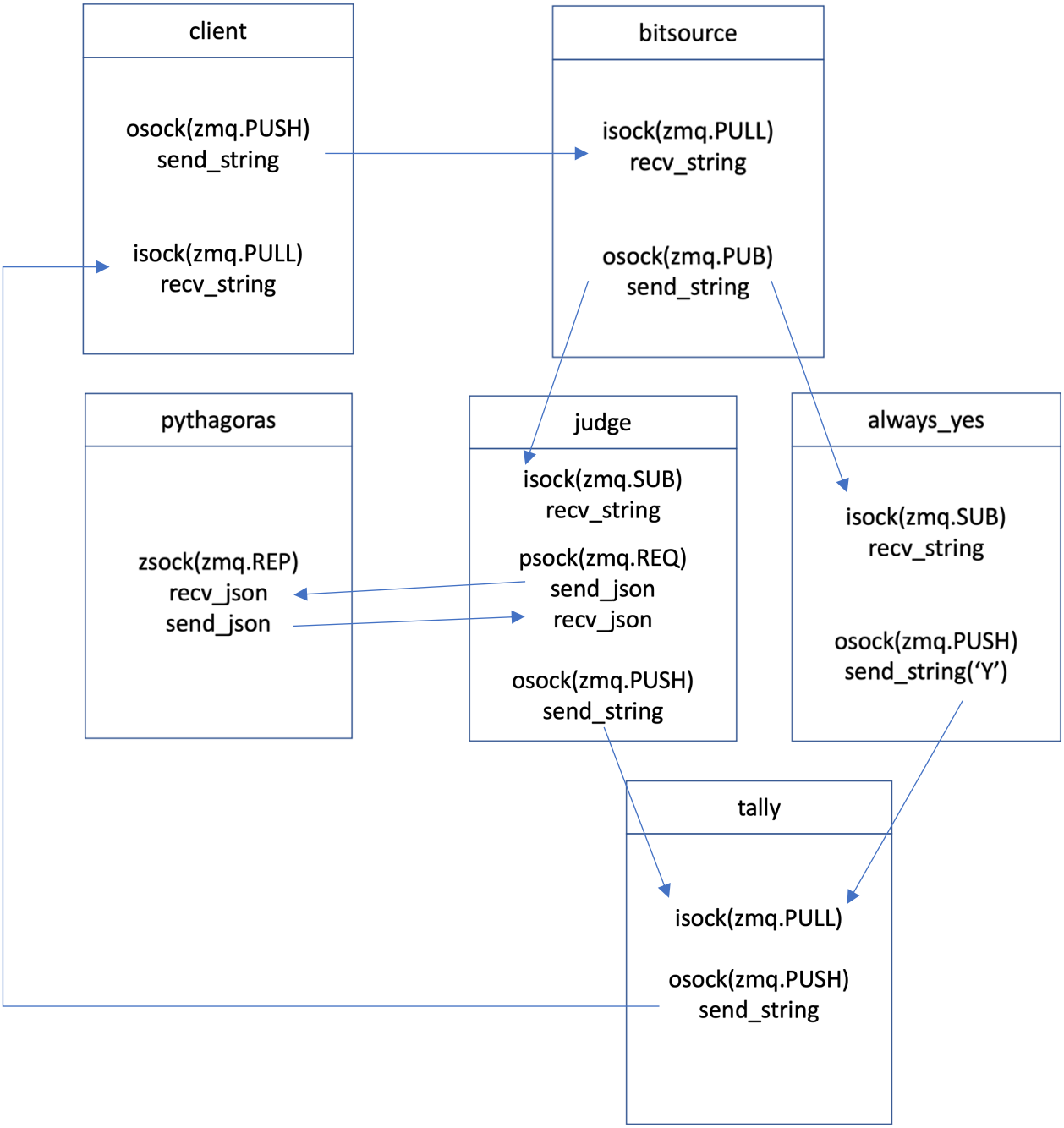
1) Logical View

Architecturally significant design:



brief description:

1. client

Its output socket is push socket to send the number of data to bitsource. Its input socket is pull socket to receive the number of iterations and pi value from tally.

First of all, It receives the number of data from real users using input function and then sends it to bitsource. Next, repeating while loop, It receives the number of iterations and pi value from tally and print it.

2. bitsource

Its input socket is pull socket to receive the number of data from client. Its output socket is publisher socket to publish random points to judge and always_yes who are subscribers.

First of all, receive the number of data from client and repeats ‘for’ loop as many data as it does. In the ‘for’ loop, as using ones_and_zeros function, It makes binary random bits and sends them to subscribers like judge and always_yes.

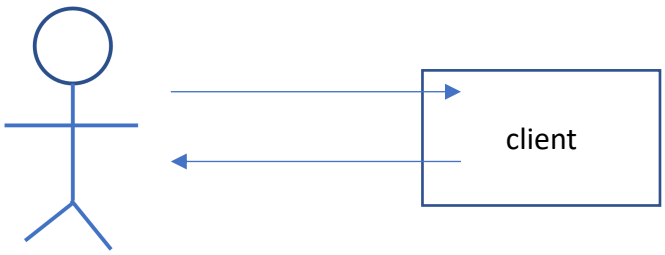
3. always_yes
Its input socket is subscriber socket to subscribe publisher, bitsource. Its output socket is push socket to send ‘Y’ or ‘N’ to tally.
First of all, set the subscriber option for b’00’ that always indicates ‘Y’ in isock. Next, repeating while loop, It receives data from bitsource and sends ‘Y’ to tally.

4. judge
It has 3socket. Its input socket is subscriber socket to subscribe publisher, bitsource. Its output socket is push socket to send ‘Y’ or ‘N’ to tally. Last socket is request socket to request pythagoras to calculate received random bits.
First of all, set the subscriber option for b‘01’, b‘10’, b‘11’ in isock. Next, repeating while loop, It receives data from bitsource, sends data to pythagoras to request and then receive reply from pythagoras. Compare number received from pythagoras with 2**(32*2). Decide ‘Y’ or ‘N’ and send them to tally.

5. pythagoras
It has one reply socket to reply judge’s request.
Repeating while loop, It receives number from judge and calculate square of number. Send it to judge.

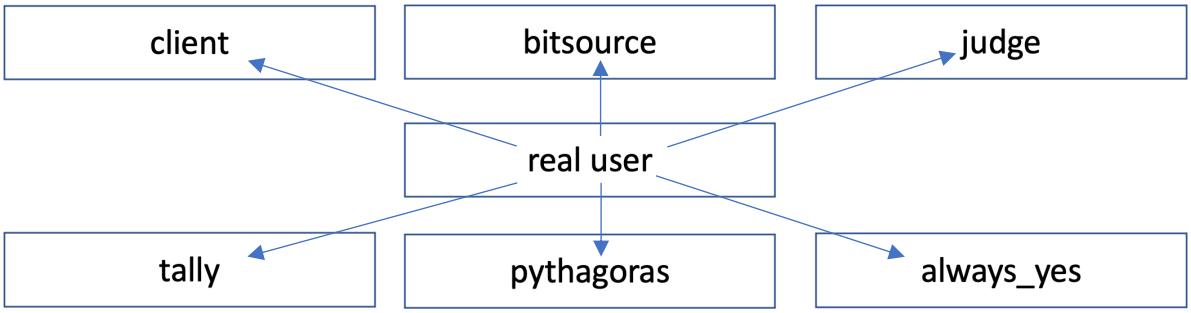
6. tally
Its input socket is pull socket to receive decision(Y or N) from judge and always_yes. Its output socket is push socket to send number of iteration and pi value to client.
Repeating while loop, receive decision from judge and always_yes. According to decision, It calculates pi value and send it to client.

use case:



The real users only communicate with ‘client’. They enter the number of data into ‘client’ and outputs the number of iterations and pi values through the ‘client’. Several internal different types of sockets functions, and files are unknown to the real user. Therefore, the real user can approach only client.

2) Process View



Real user means a user who executes an actual process in a terminal. The six individual workers are executed in each terminal by a real user.

Client
This process starts on the client and ends on the client. Client forms two push and pull relationships. It connects with bitsource using the fromclient port and with tally using the toclient port. It connects the socket to the bitsource to push and binds the socket to tally to pull the socket. Not only that, but it sends the socket to bitsource and delivers data. When receiving data from tally, it repeats the infinite while loop and prints that data.

Publisher and subscriber
Bitsource and always_yes, judge are publisher and subscriber relationship. They use pubsub port. Bitsource binds the publisher socket and judge, always_yes connect the sockets to subscribe. When bitsource sends random binary bits through pubsub port, always_yes receives b'00' case and judge receives b'01', b'10', b'11' cases.

Request and reply
Pythagoras and judge are request and reply relationship. They use reqrep port. Pythagoras binds the socket and judge connect the socket. Judge requests pythagoras and Pythagoras reply to judge.

Push and pull
Judge and always_yes, and tally have push and pull relationship. They use pushpull port. Judge and always_yes push the data, so they connect the socket. Tally binds the socket to pull data. Judge and always_yes sends Y or N decision to tally until there are no random bits received from bitsource. And then tally performs the next step. It connects the socket to push data to client. During while loop, it receives decision from judge and always_yes, calculates pi value and sends the number of iterations and pi value.

why use push and pull relationship between client and bitsource, client and tally?
The types of network topology learned in class are publisher-subscriber, push-pull, and request-reply.

First of all, in the relationship of publisher-subscriber, subscriber is conditioned by the type of data sent by the publisher. This means depending on the type of data, we need to process accordingly. However, there is no need to do this in the relationship between client and bitsource. Client just needs to send the input number to bitsource. The action is not determined by the number received, but the number is transmitted to the bitsource regardless of what is entered. Also in client-tally relationship, client only prints the data sent to the tally. The destination does not depend on the data sent by tally to the client.

Second, the request-reply relationship is similar to the server and client relationship. Just as the server responds when a client makes a request, when a request socket is sent, the receiver performs some work and then reply. In other words, it is a relationship that requests and receives responses for certain tasks. However, the relationship between client and bitsource, between client and tally does not require performing a specific task.

For these reasons, I chose a push-pull relationship rather than publisher-subscriber and request-reply relationships.

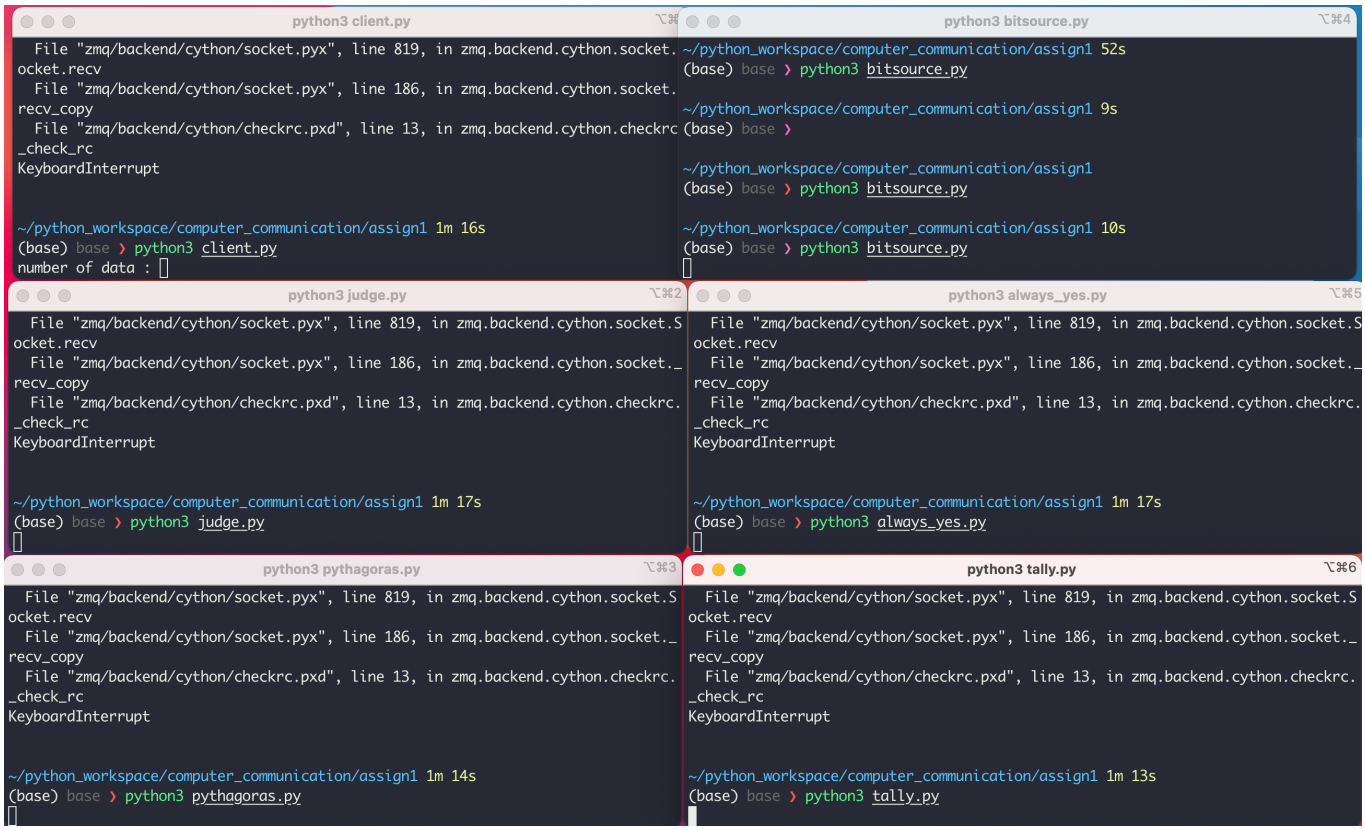
3) High-level Design

1. client sends the number of data to bitsource.
2. bitsource sends the random bits to judge and always_yes.
3. judge and always_yes send decision to tally.
4. judge sends some part of random bits to pythagoras.
5. pythagoras sends square of number to judge.
6. tally sends the number of iterations and pi value to client.

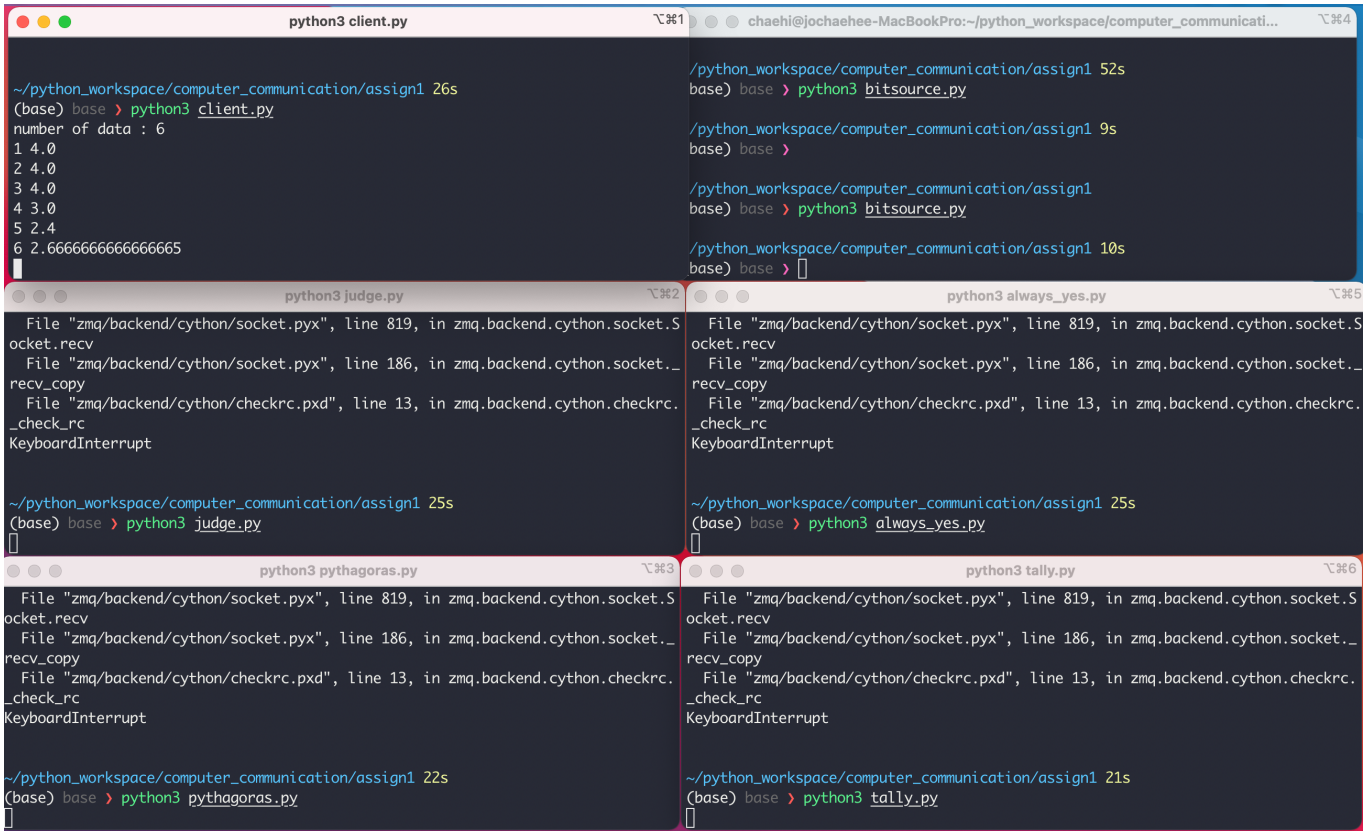
4) Test Case:

1. enter 6

Initially, this is a screen where 6 users are run from independent terminals.
Waiting for the client file to enter the number of data.

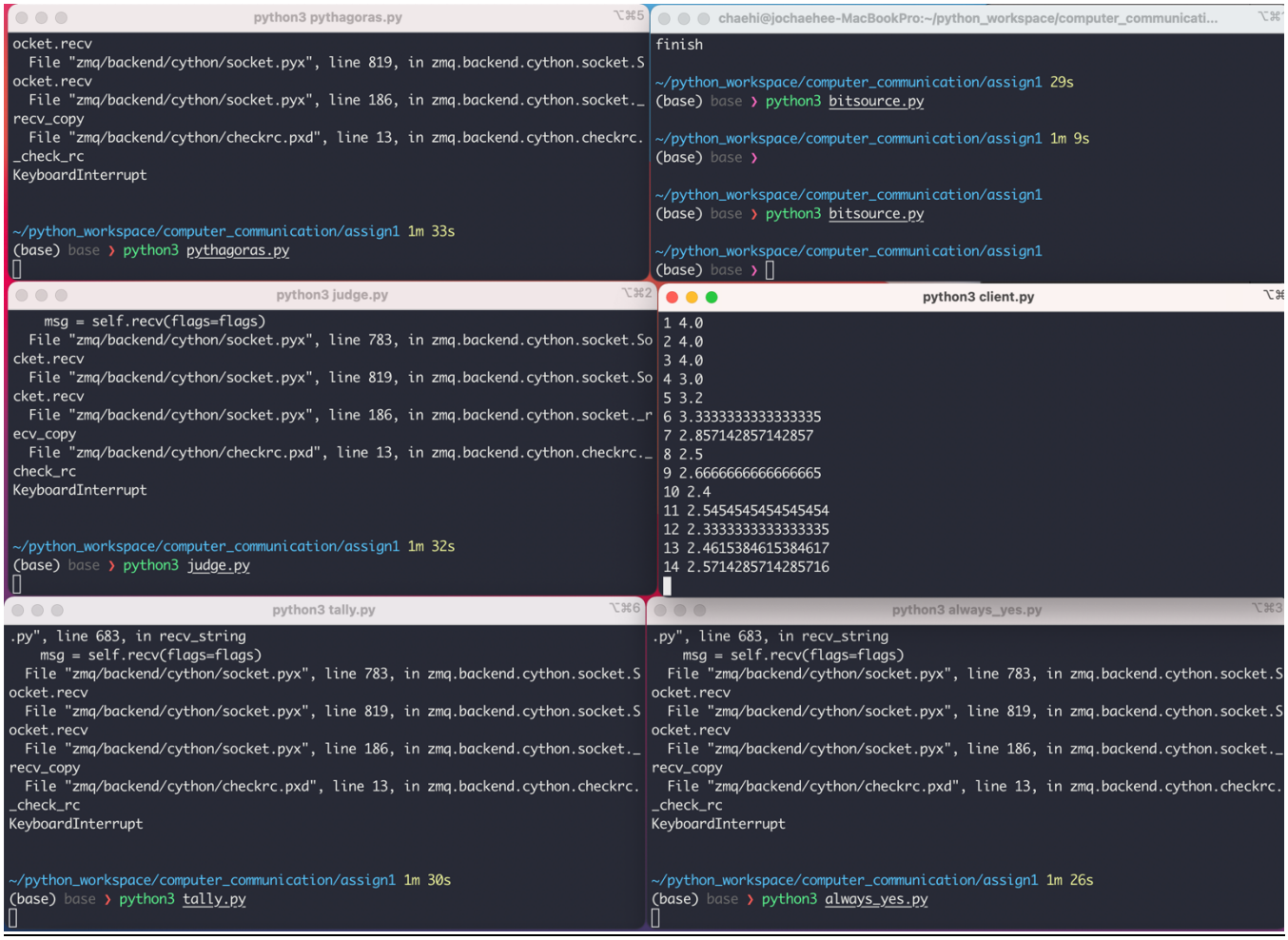


After enter 6 in client,

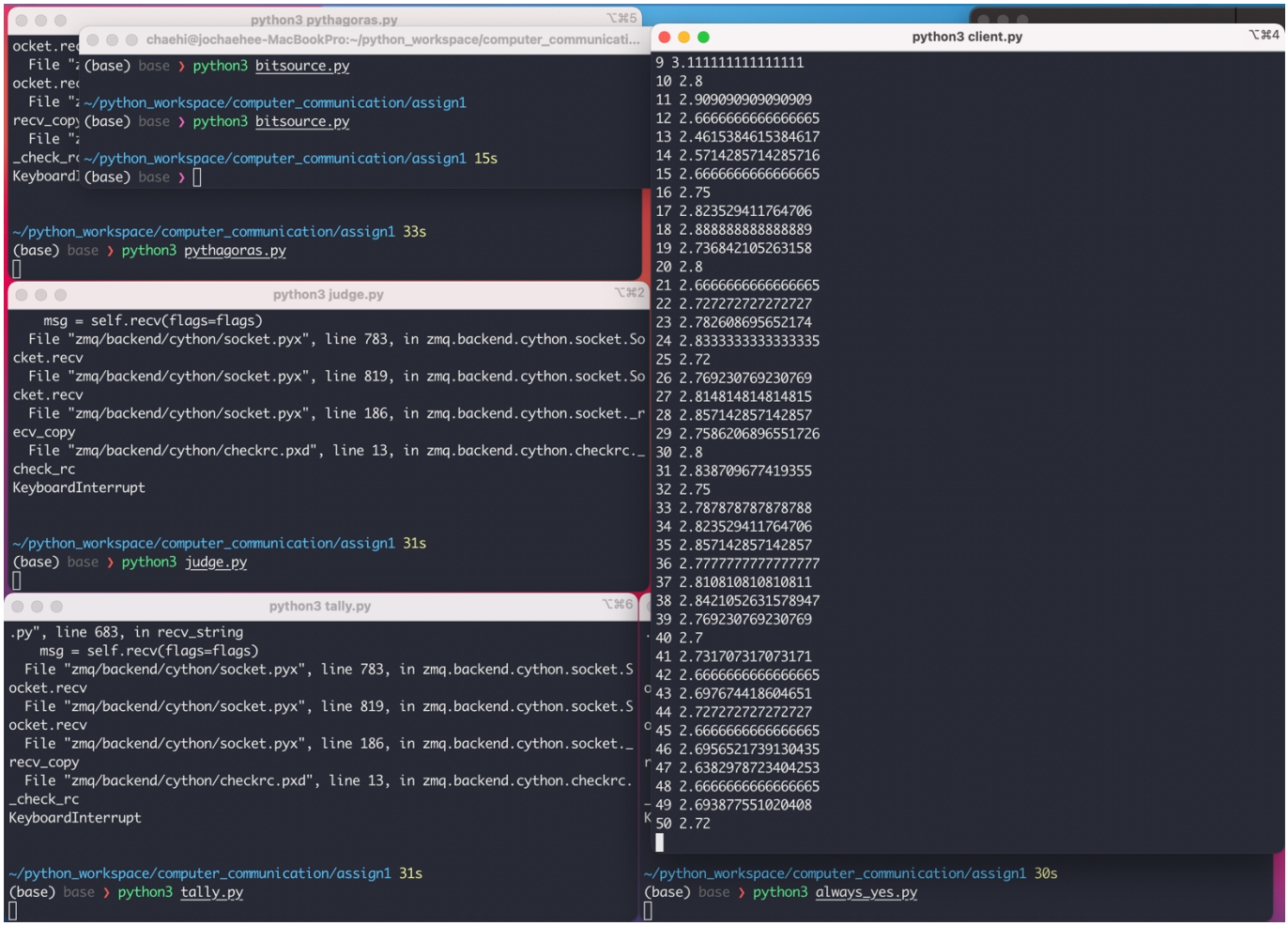


client prints the number of iterations and pi value

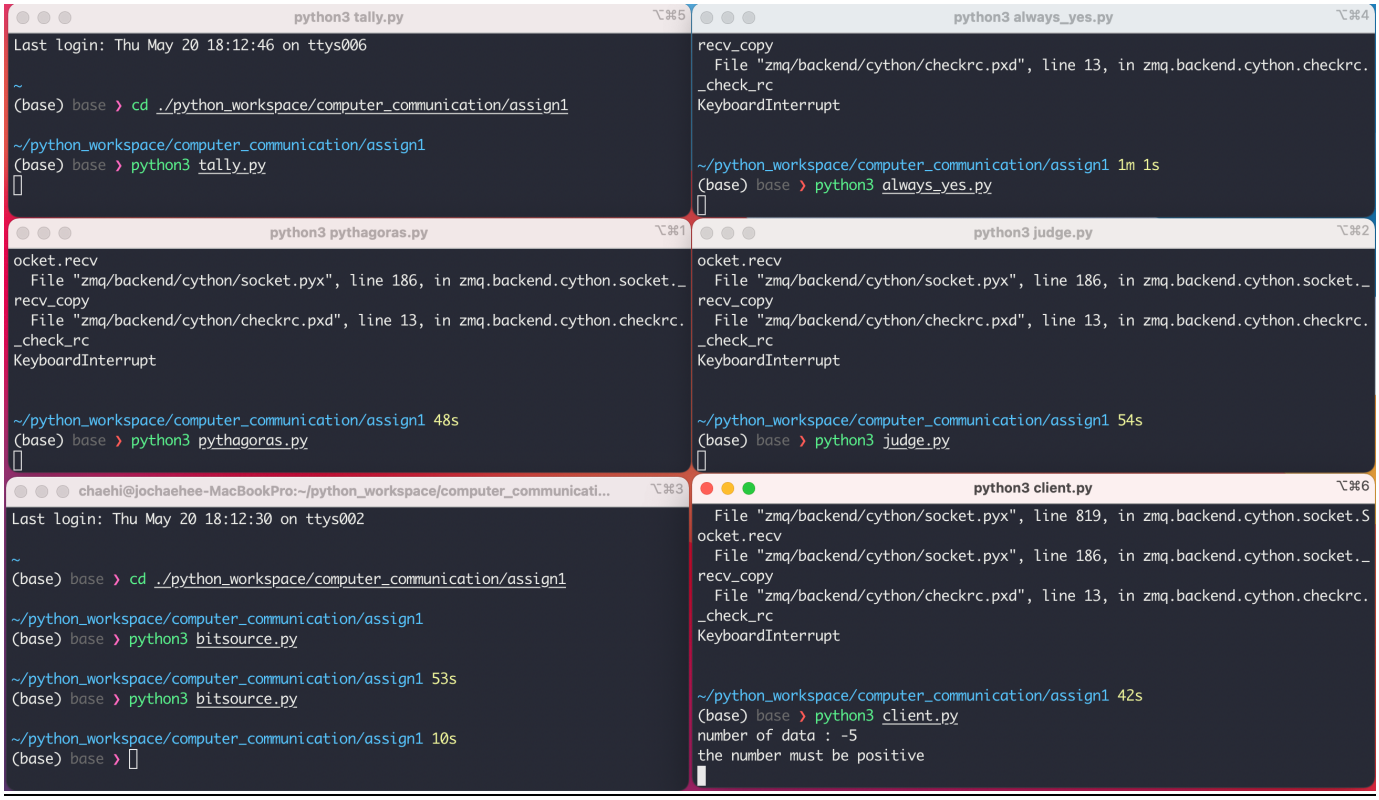
2. enter 14


python3 pythagoras.py
ocket.recv
File "zmq/backend/cython/socket.pyx", line 819, in zmq.backend.cython.socket.S
ocket.recv
File "zmq/backend/cython/socket.pyx", line 186, in zmq.backend.cython.socket._
recv_copy
File "zmq/backend/cython/checkrc.pxd", line 13, in zmq.backend.cython.checkrc.
_check_rc
KeyboardInterrupt
~/python_workspace/computer_communication/assign1 1m 33s
(base) base ❯ python3 pythagoras.py
chaehi@jochaehee-MacBookPro:~/python_workspace/computer_communicati...
finish
~/python_workspace/computer_communication/assign1 29s
(base) base ❯ python3 bitsource.py
~/python_workspace/computer_communication/assign1 1m 9s
(base) base ❯
~/python_workspace/computer_communication/assign1
(base) base ❯ python3 bitsource.py
~/python_workspace/computer_communication/assign1
(base) base ❯
python3 judge.py
msg = self.recv(flags=flags)
File "zmq/backend/cython/socket.pyx", line 783, in zmq.backend.cython.socket.So
cket.recv
File "zmq/backend/cython/socket.pyx", line 819, in zmq.backend.cython.socket.So
cket.recv
File "zmq/backend/cython/socket.pyx", line 186, in zmq.backend.cython.socket._r
ecv_copy
File "zmq/backend/cython/checkrc.pxd", line 13, in zmq.backend.cython.checkrc._
check_rc
KeyboardInterrupt
~/python_workspace/computer_communication/assign1 1m 32s
(base) base ❯ python3 judge.py
python3 client.py
1 4.0
2 4.0
3 4.0
4 3.0
5 3.2
6 3.3333333333333335
7 2.857142857142857
8 2.5
9 2.6666666666666665
10 2.4
11 2.5454545454545454
12 2.3333333333333335
13 2.4615384615384617
14 2.5714285714285716
python3 tally.py
.py", line 683, in recv_string
msg = self.recv(flags=flags)
File "zmq/backend/cython/socket.pyx", line 783, in zmq.backend.cython.socket.S
ocket.recv
File "zmq/backend/cython/socket.pyx", line 819, in zmq.backend.cython.socket.S
ocket.recv
File "zmq/backend/cython/socket.pyx", line 186, in zmq.backend.cython.socket._
recv_copy
File "zmq/backend/cython/checkrc.pxd", line 13, in zmq.backend.cython.checkrc.
_check_rc
KeyboardInterrupt
~/python_workspace/computer_communication/assign1 1m 30s
(base) base ❯ python3 tally.py
python3 always_yes.py
.py", line 683, in recv_string
msg = self.recv(flags=flags)
File "zmq/backend/cython/socket.pyx", line 783, in zmq.backend.cython.socket.S
ocket.recv
File "zmq/backend/cython/socket.pyx", line 819, in zmq.backend.cython.socket.S
ocket.recv
File "zmq/backend/cython/socket.pyx", line 186, in zmq.backend.cython.socket._
recv_copy
File "zmq/backend/cython/checkrc.pxd", line 13, in zmq.backend.cython.checkrc.
_check_rc
KeyboardInterrupt
~/python_workspace/computer_communication/assign1 1m 26s
(base) base ❯ python3 always_yes.py

3. enter 50


python3 pythagoras.py
chaehi@jochaehee-MacBookPro:~/python_workspace/computer_communicati...
python3 client.py
python3 judge.py
python3 tally.py
(base) base ❯ python3 bitsource.py
~/python_workspace/computer_communication/assign1
(base) base ❯ python3 bitsource.py
~/python_workspace/computer_communication/assign1 15s
(base) base ❯
~/python_workspace/computer_communication/assign1 33s
(base) base ❯ python3 pythagoras.py
msg = self.recv(flags=flags)
File "zmq/backend/cython/socket.pyx", line 783, in zmq.backend.cython.socket.Socket.recv
File "zmq/backend/cython/socket.pyx", line 819, in zmq.backend.cython.socket.Socket.recv
File "zmq/backend/cython/socket.pyx", line 186, in zmq.backend.cython.socket._recv_copy
File "zmq/backend/cython/checkrc.pxd", line 13, in zmq.backend.cython.checkrc._check_rc
KeyboardInterrupt
~/python_workspace/computer_communication/assign1 31s
(base) base ❯ python3 judge.py
.py", line 683, in recv_string
msg = self.recv(flags=flags)
File "zmq/backend/cython/socket.pyx", line 783, in zmq.backend.cython.socket.Socket.recv
File "zmq/backend/cython/socket.pyx", line 819, in zmq.backend.cython.socket.Socket.recv
File "zmq/backend/cython/socket.pyx", line 186, in zmq.backend.cython.socket._recv_copy
File "zmq/backend/cython/checkrc.pxd", line 13, in zmq.backend.cython.checkrc._check_rc
KeyboardInterrupt
~/python_workspace/computer_communication/assign1 31s
(base) base ❯ python3 tally.py
9 3.111111111111111
10 2.8
11 2.909090909090909
12 2.6666666666666665
13 2.4615384615384617
14 2.5714285714285716
15 2.6666666666666665
16 2.75
17 2.823529411764706
18 2.888888888888889
19 2.736842105263158
20 2.8
21 2.6666666666666665
22 2.727272727272727
23 2.782608695652174
24 2.8333333333333335
25 2.72
26 2.769230769230769
27 2.814814814814815
28 2.857142857142857
29 2.7586206896551726
30 2.8
31 2.838709677419355
32 2.75
33 2.787878787878788
34 2.823529411764706
35 2.857142857142857
36 2.7777777777777777
37 2.810810810810811
38 2.8421052631578947
39 2.769230769230769
40 2.7
41 2.731707317073171
42 2.6666666666666665
43 2.697674418604651
44 2.727272727272727
45 2.6666666666666665
46 2.6956521739130435
47 2.6382978723404253
48 2.6666666666666665
49 2.693877551020408
50 2.72
~/python_workspace/computer_communication/assign1 30s
(base) base ❯ python3 always_yes.py

4. enter negative number



exception handling: when the number of data is negative number.