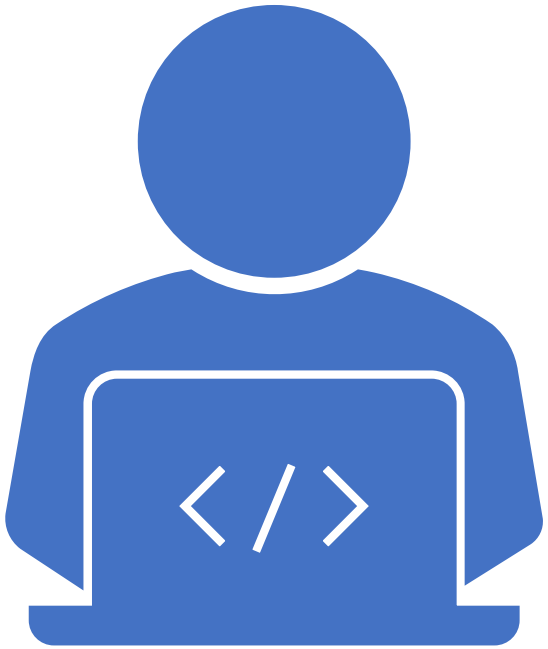
CAN 201

Dr. Fei Cheng & Dr. Gordon Boateng

Lab 6
Parallel computing & Data Structure

…



- IF YOU HAVE NOT COMPLETED THE TASK IN LAB5, PLEASE TRY TO COMPLETE SOON AND LET TA CHECK WHAT YOU DID.

Contents

- Parallel computing + TCP multiple clients support
- Data structure
- struct module

Parallel computing

- How to use parallel computing in python?
  - ***multiprocessing*** module – process-based parallelism
  - ***threading*** module – thread-based parallelism
- How to support multi-clients for TCP?
  - Using ***threading***

Use Process class - *multiprocessing*

- Try the following example:

```
from multiprocessing import Process
import time


def one_process(name):
    for i in range(10):
        print(name, i)
        time.sleep(0.5)


if __name__ == '__main__':
    p = Process(target=one_process, args=('In sub process',))
    p.start()
    time.sleep(0.2)
    for i in range(5):
        print('In main process', i)
        time.sleep(1)
    p.join()
```

Use Pool class - *multiprocessing*

- Try the following example:

```
from multiprocessing.pool import Pool
from multiprocessing import cpu_count


def one_function(i):
    return i


if __name__ == '__main__':

    pool = Pool(cpu_count() - 1)

    r = pool.map(one_function, (1, 'a', 0.5))
    print(r)
```

Please find more information @ https://docs.python.org/3.9/library/multiprocessing.html

threading Module

- Higher-level threading interfaces on top of the lower level ***_thread*** module
- Try the following example:

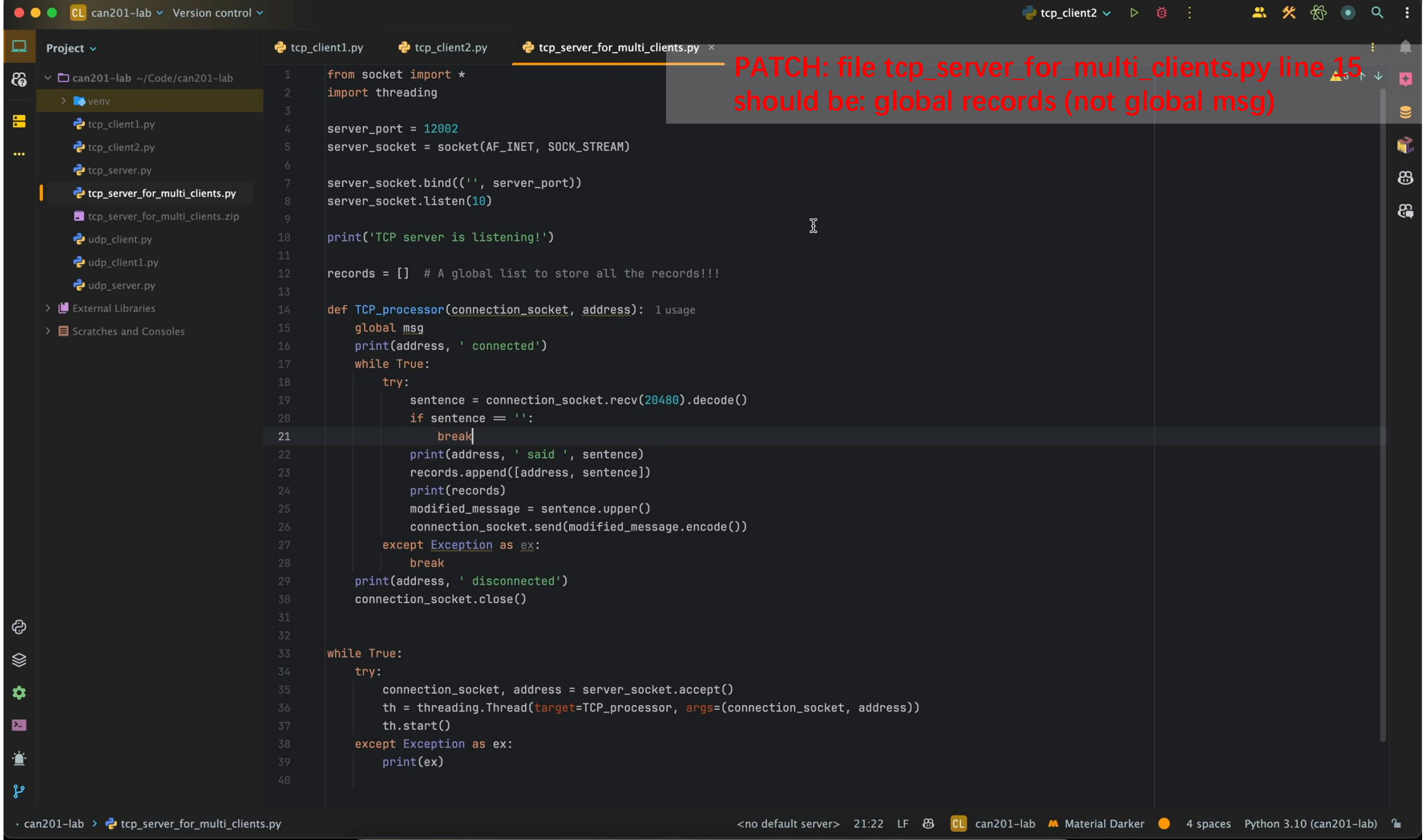
```
from threading import Thread
import time


def one_thread(name):
    for i in range(10):
        print(name, i)
        time.sleep(0.5)


if __name__ == '__main__':
    p = Thread(target=one_thread, args=('In thread',))
    p.start()
    time.sleep(0.2)
    for i in range(5):
        print('In main process', i)
        time.sleep(1)
    p.join()
```

How to support multi-clients for TCP

- Code: https://box.xjtlu.edu.cn/f/9f13a69f537143fca29e/

- Video: https://box.xjtlu.edu.cn/f/3f46a45e9daf4d90a9f5/

For UDP… UDP can support multi-clients directly.

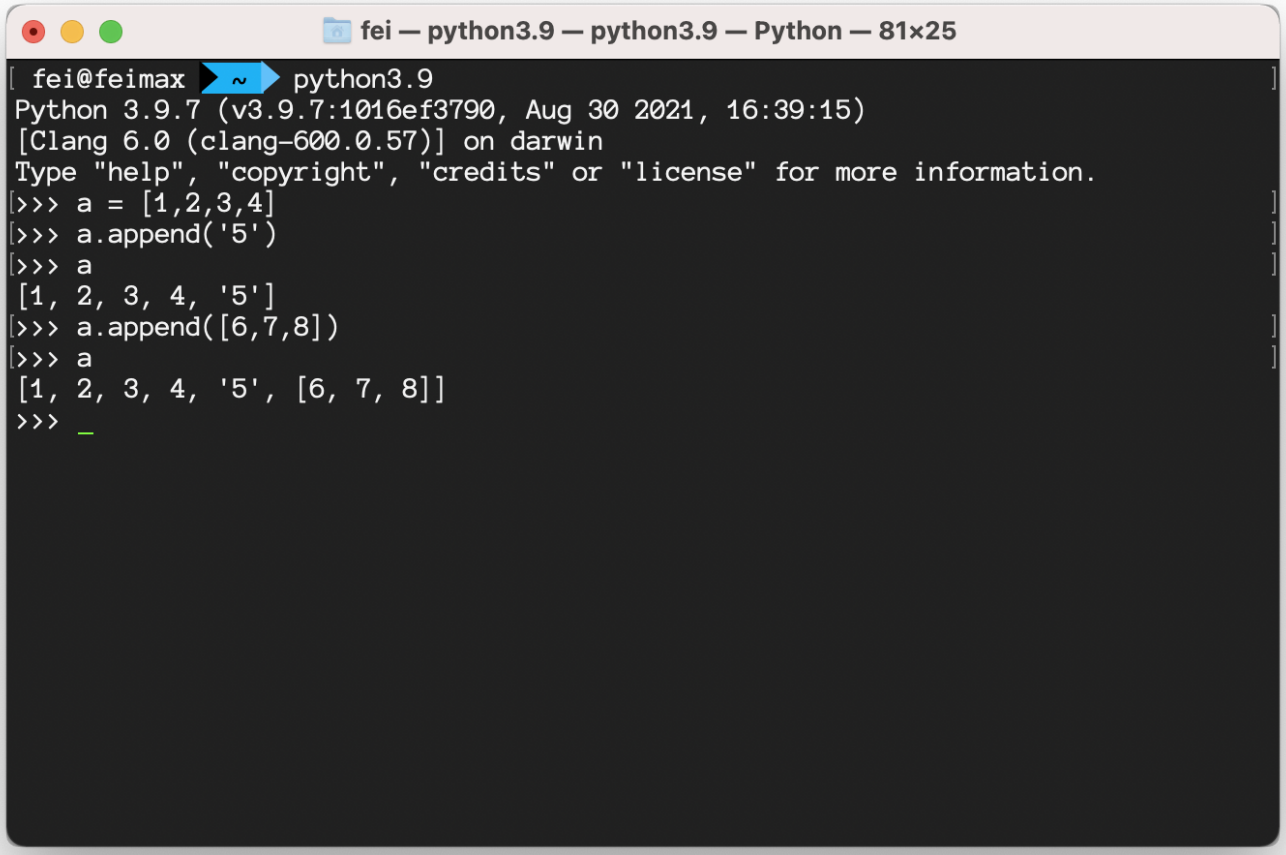

can201-lab
Version control
tcp_client2
Project
tcp_client1.py
tcp_client2.py
tcp_server_for_multi_clients.py
PATCH: file tcp_server_for_multi_clients.py line 15
should be: global records (not global msg)
can201-lab ~/Code/can201-lab
venv
tcp_client1.py
tcp_client2.py
tcp_server.py
tcp_server_for_multi_clients.py
tcp_server_for_multi_clients.zip
udp_client.py
udp_client1.py
udp_server.py
External Libraries
Scratches and Consoles
from socket import *
import threading
server_port = 12002
server_socket = socket(AF_INET, SOCK_STREAM)
server_socket.bind(('', server_port))
server_socket.listen(10)
print('TCP server is listening!')
records = [] # A global list to store all the records!!!
def TCP_processor(connection_socket, address): 1 usage
global msg
print(address, ' connected')
while True:
try:
sentence = connection_socket.recv(20480).decode()
if sentence == '':
break
print(address, ' said ', sentence)
records.append([address, sentence])
print(records)
modified_message = sentence.upper()
connection_socket.send(modified_message.encode())
except Exception as ex:
break
print(address, ' disconnected')
connection_socket.close()
while True:
try:
connection_socket, address = server_socket.accept()
th = threading.Thread(target=TCP_processor, args=(connection_socket, address))
th.start()
except Exception as ex:
print(ex)
can201-lab
tcp_server_for_multi_clients.py
<no default server>
21:22
LF
can201-lab
Material Darker
4 spaces
Python 3.10 (can201-lab)

Data structure

- List
- Tuple
- Set(optional)
- Dictionary and JSON
- How to handle binary data – ***struct*** module

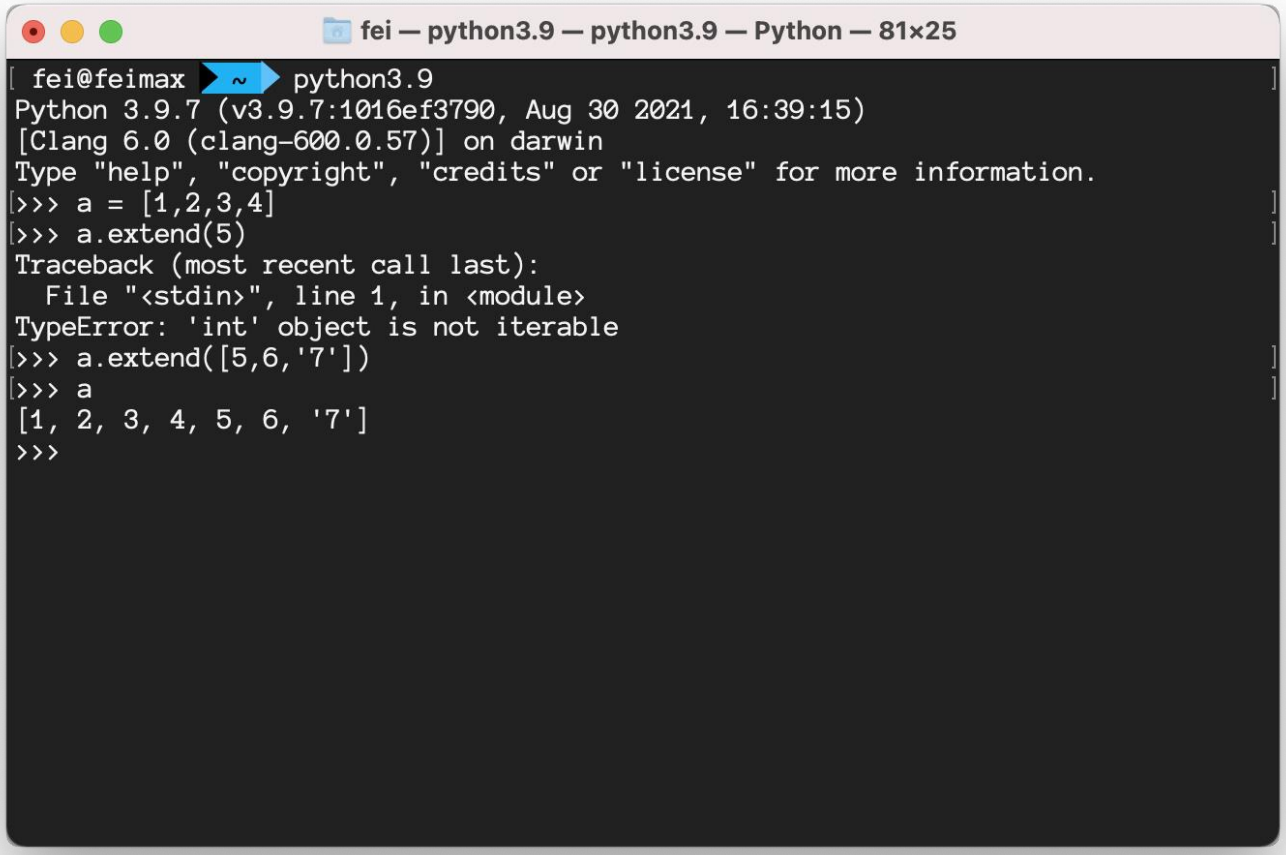
List

- Review
  - `list.append(x)`



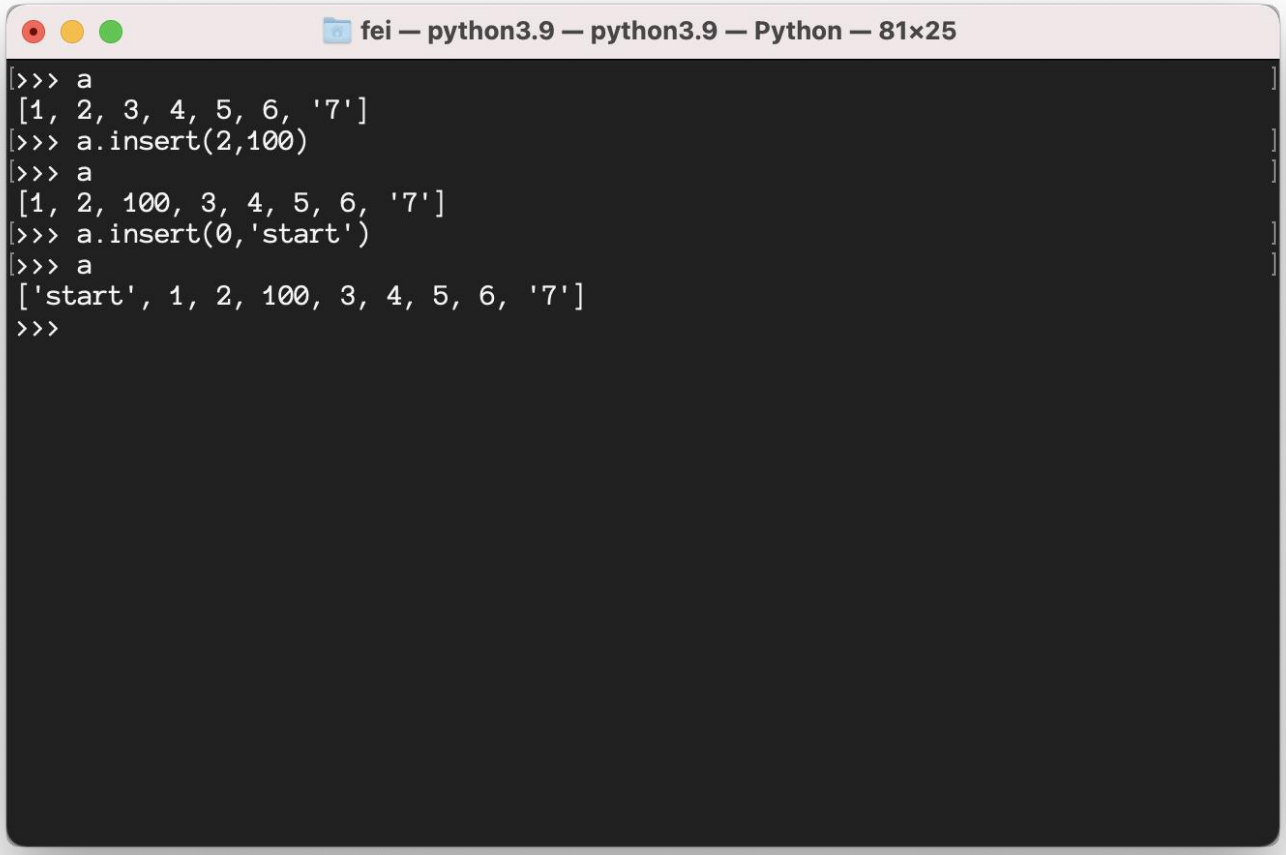
List

- Review
  - `list.append(x)`
  - `list.extend(iterable)`



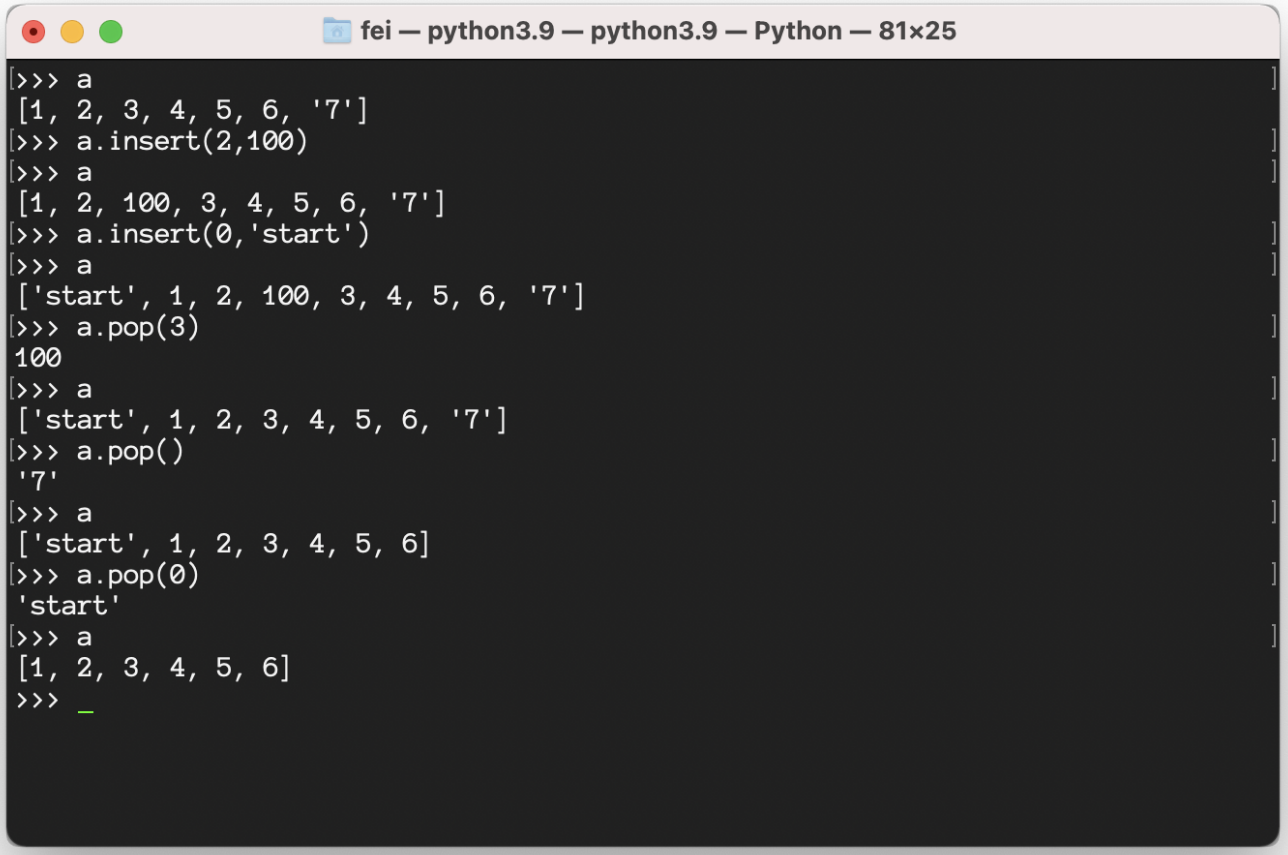
List

- Review
  - `list.append(x)`
  - `list.extend(iterable)`
  - `list.insert(i, x)`



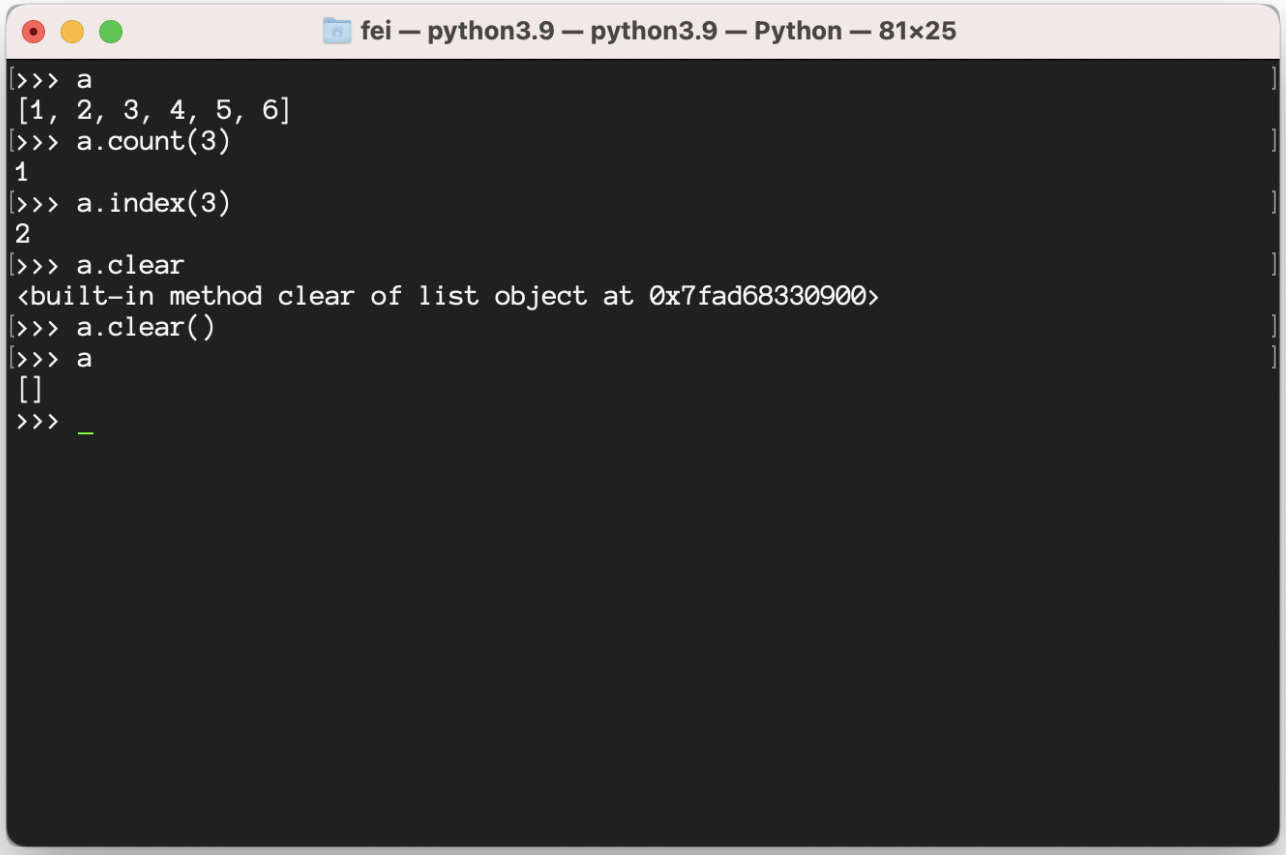
List

- Review
  - `list.append(x)`
  - `list.extend(iterable)`
  - `list.insert(i, x)`
  - `list.pop(i)`



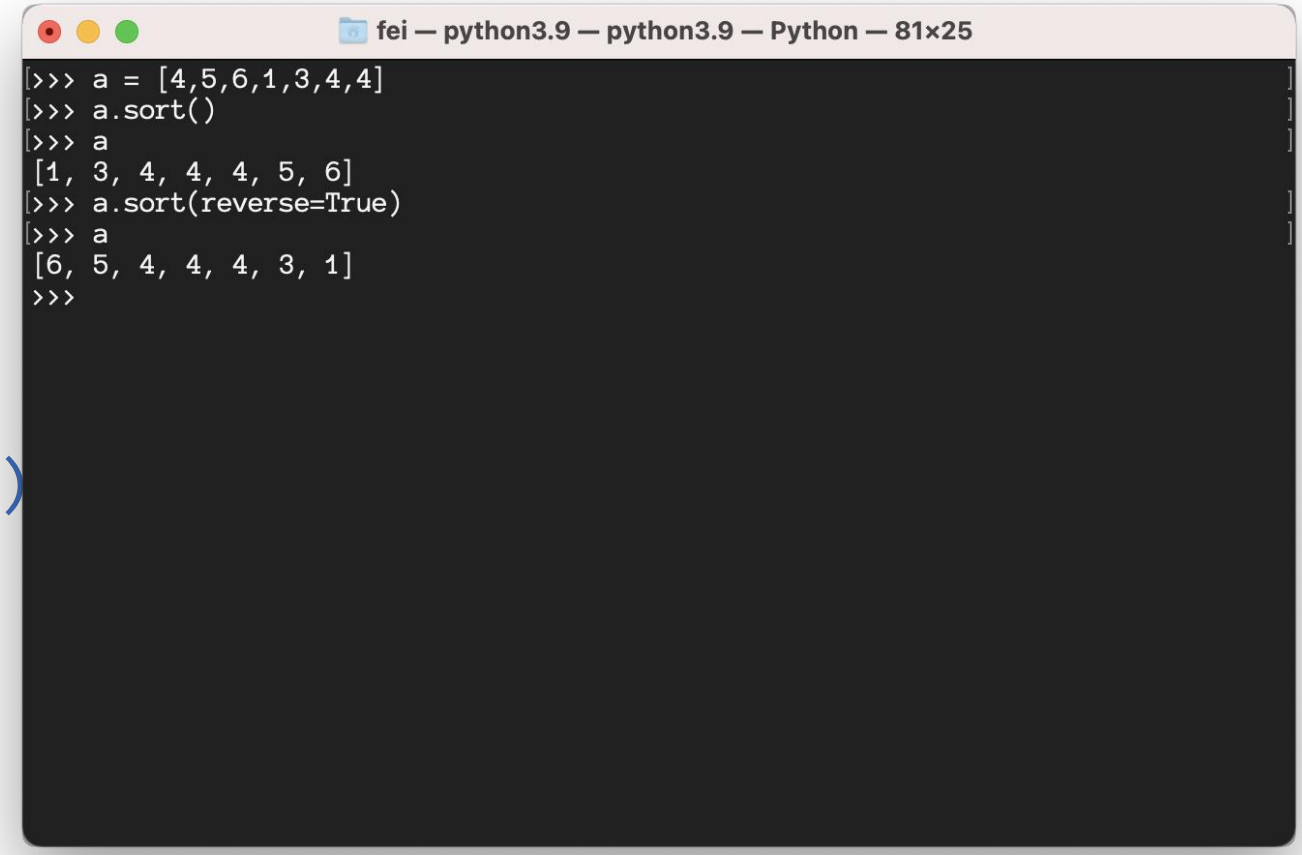
List

- Review
  - `list.append(x)`
  - `list.extend(iterable)`
  - `list.insert(i, x)`
  - `list.pop([x])`
  - `list.count(x)`



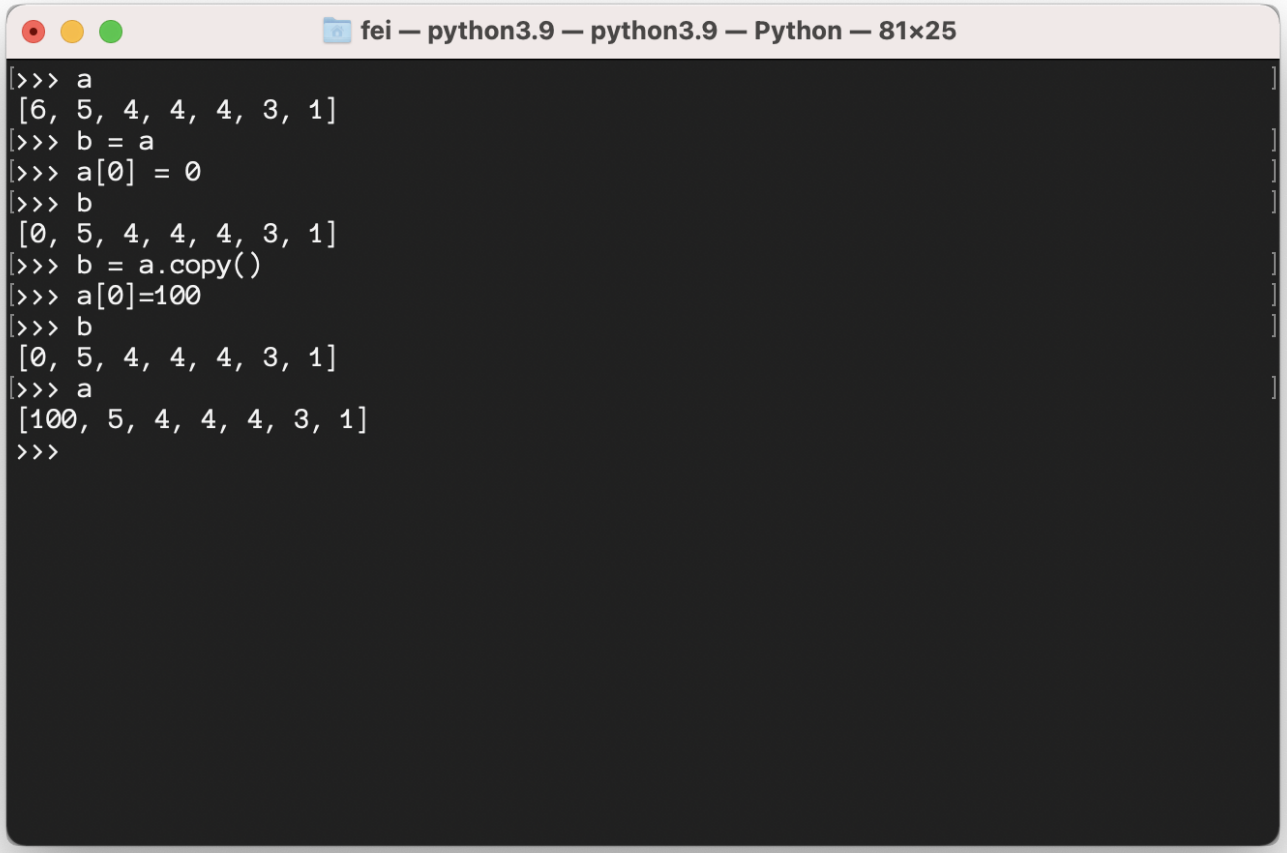
List

- Review
  - `list.append(x)`
  - `list.extend(iterable)`
  - `list.insert(i, x)`
  - `list.pop([x])`
  - `list.count(x)`
  - `list.sort([key, rverse])`



List

- Review
  - `list.append(x)`
  - `list.extend(iterable)`
  - `list.insert(i, x)`
  - `list.pop([x])`
  - `list.count(x)`
  - `list.sort([key, rverse])`
  - `list.copy()`



List Comprehensions

```
squares = []
for x in range(10):
    squares.append(x**2)
```

```
squares = list(map(lambda x: x**2, range(10)))
```

```
squares = [x**2 for x in range(10)]
```

List Comprehensions

```
squares = []
for x in range(10):
    squares.append(x**2
```

```
squares = list(map(lambd
```

```
squares = [x**2 for x in
```

Tuple

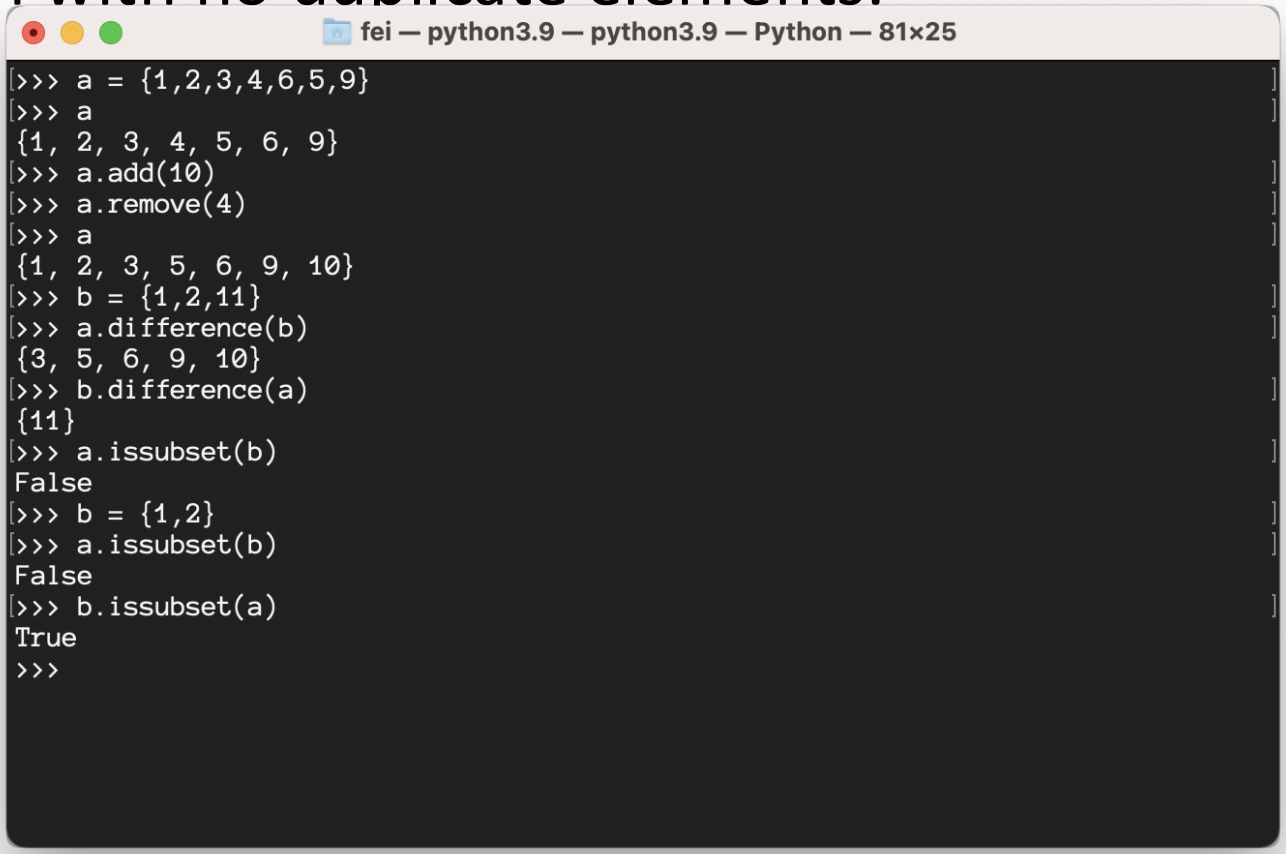
- Another sequence data type.
- `t = (1, 2, '4', 'string')`
- Sequence data types: indexing,slicing … operations
  - list
  - tuple
  - range
- An immutable list

Set (optional)

- A set is an unordered collection with no duplicate elements.
- `set.add(x)`
- `set.remove(x)`
- `set.difference(s)`
- `set.intersection(s)`
- `set.issubset(s)`

Set (optional)

- A set is an unordered collection with no duplicate elements.
- `set.add(x)`
- `set.remove(x)`
- `set.difference(s)`
- `set.intersection(s)`
- `set.issubset(s)`



IMPORTANT

Dictionary

- Dictionaries are indexed by *keys*
- Keys can be numbers or strings
- key - value pairs
- First example:

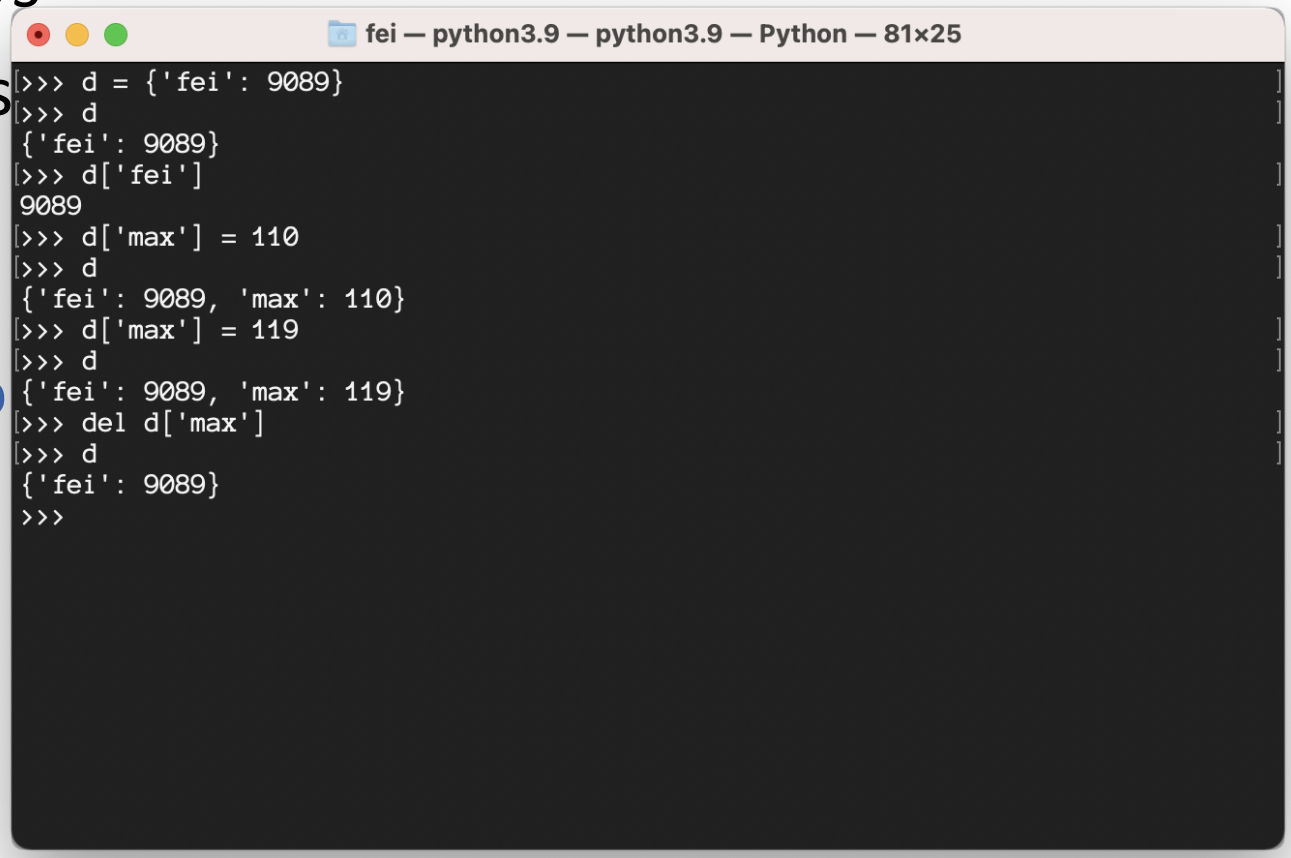
```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
```

IMPORTANT

Dictionary

- Dictionaries are indexed by *keys*
- Keys can be numbers or strings
- key - value pairs
- First example:

```
tel = {'jack': 4098, 'sap
tel['guido'] = 4127
```



IMPORTANT

Dictionary

- How to know the key exists or not?

IMPORTANT

Dictionary

- How to enumerate the dict?

IMPORTANT

Dictionary

- How to send a dict using socket?
  - dict - > string->binary data
  - binary data -> string -> dict

  - JSON!

IMPORTANT

JSON

- JavaScript Object Notation
- dict <-> JSON formatted string

JSON

IMPORTANT

- JavaScript Object Notation
- dict <-> JSON

struct module

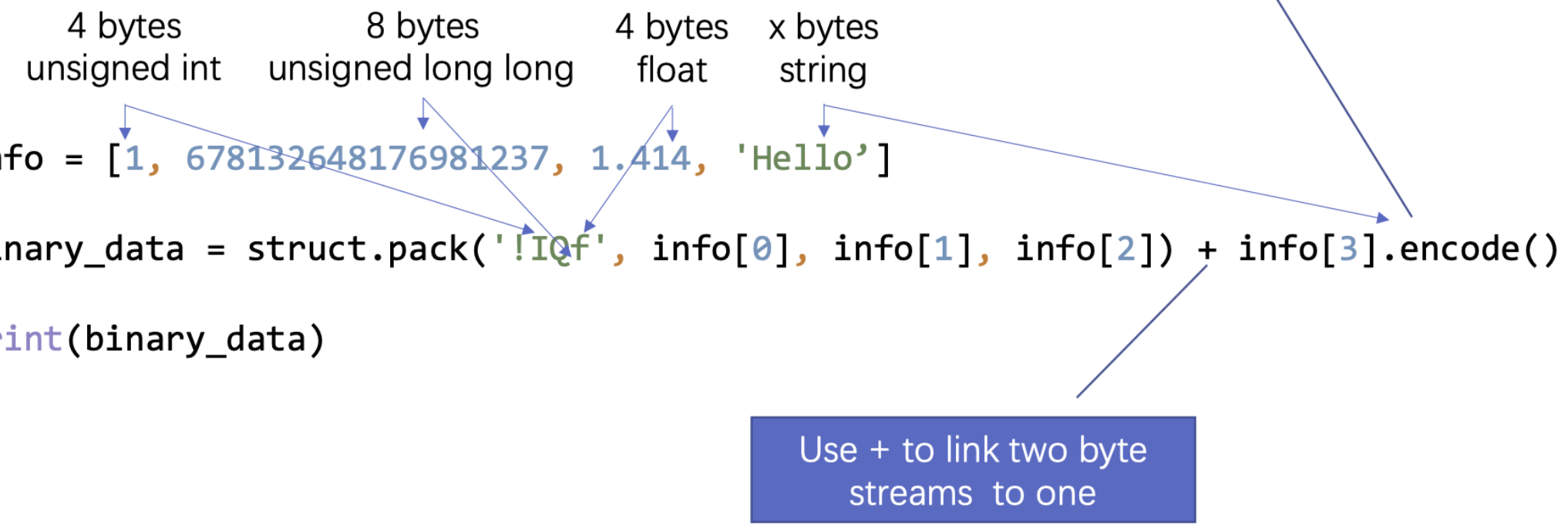
ATTENTION PLEASE!!!

- *struct* module can be used in handling binary data stored in files or from network connections
- *struct* module performs conversions between Python values and C structs represented as Python bytes (we don't use this feature in this project)
- Review:
  - In the last lab, we need to use string.encode() to convert string to bytes
  - How to send ***int or float?***

struct module

As the length of a string is not fixed, we put it to the end. Think: how to add two strings

- Assume:
  - We want to send [1, 67813264817698123 7, 1.414, ‘Hello’] using socket



struct module

https://en.wikipedia.org/wiki/Endianness

```
# Get data back:

v_i, v_Q, v_f = struct.unpack('!IQf', binary_data[:16])

v_str = binary_data[16:].decode()

print(v_i, v_Q, v_f, v_str)
```

See “make_packet” and “get_tcp_packet” functions in the server.py to get more ideas about how to deal with number and binary data together!

Format	C Type	Python type	Standard size	Notes
`x`	pad byte	no value		
`c`	`char`	bytes of length 1	1	
`b`	`signed char`	integer	1	(1), (2)
`B`	`unsigned char`	integer	1	(2)
`?`	`_Bool`	bool	1	(1)
`h`	`short`	integer	2	(2)
`H`	`unsigned short`	integer	2	(2)
`i`	`int`	integer	4	(2)
`I`	`unsigned int`	integer	4	(2)
`l`	`long`	integer	4	(2)
`L`	`unsigned long`	integer	4	(2)
`q`	`long long`	integer	8	(2)
`Q`	`unsigned long long`	integer	8	(2)
`n`	`ssize_t`	integer		(3)
`N`	`size_t`	integer		(3)
`e`	(6)	float	2	(4)
`f`	`float`	float	4	(4)
`d`	`double`	float	8	(4)
`s`	`char[]`	bytes		
`p`	`char[]`	bytes		
`P`	`void *`	integer		(5)

Thanks