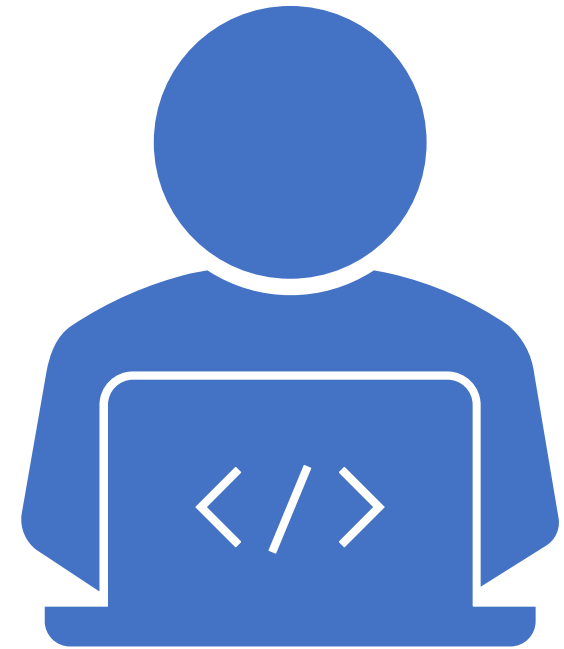


CAN 201

Dr. Fei Cheng

Lab 5
Parallel computing &
Data Structure &
File operation &

...



Demo for Practice of last week

<https://box.xjtlu.edu.cn/f/0489e9fb3a5145ed83da/>



Contents

- **Parallel computing**
- **Data structure**
- **struct module**
- **File operation**

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/f7216f97335c45ab8650/?dl=1>
 - PATCH: file tcp_server_for_multi_clients.py line 15 should be: global records (not global msg)

```
def TCP_processor(connection_socket, address): 1 usage
    global records
    print(address, ' connected')
```

- Video: <https://box.xjtlu.edu.cn/f/54de17ea4b814955b818/>



For UDP... UDP can support multi-clients directly.

Project 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

1 from socket import *
2 import threading
3
4 server_port = 12002
5 server_socket = socket(AF_INET, SOCK_STREAM)
6
7 server_socket.bind(('', server_port))
8 server_socket.listen(10)
9
10 print('TCP server is listening!')
11
12 records = [] # A global list to store all the records!!!
13
14 def TCP_processor(connection_socket, address): 1 usage
15 global msg
16 print(address, ' connected')
17 while True:
18 try:
19 sentence = connection_socket.recv(20480).decode()
20 if sentence == '':
21 break
22 print(address, ' said ', sentence)
23 records.append([address, sentence])
24 print(records)
25 modified_message = sentence.upper()
26 connection_socket.send(modified_message.encode())
27 except Exception as ex:
28 break
29 print(address, ' disconnected')
30 connection_socket.close()
31
32
33 while True:
34 try:
35 connection_socket, address = server_socket.accept()
36 th = threading.Thread(target=TCP_processor, args=(connection_socket, address))
37 th.start()
38 except Exception as ex:
39 print(ex)
40

A patch:

- file tcp_server_for_multi_clients.py line 15 should be: global records (not global msg)

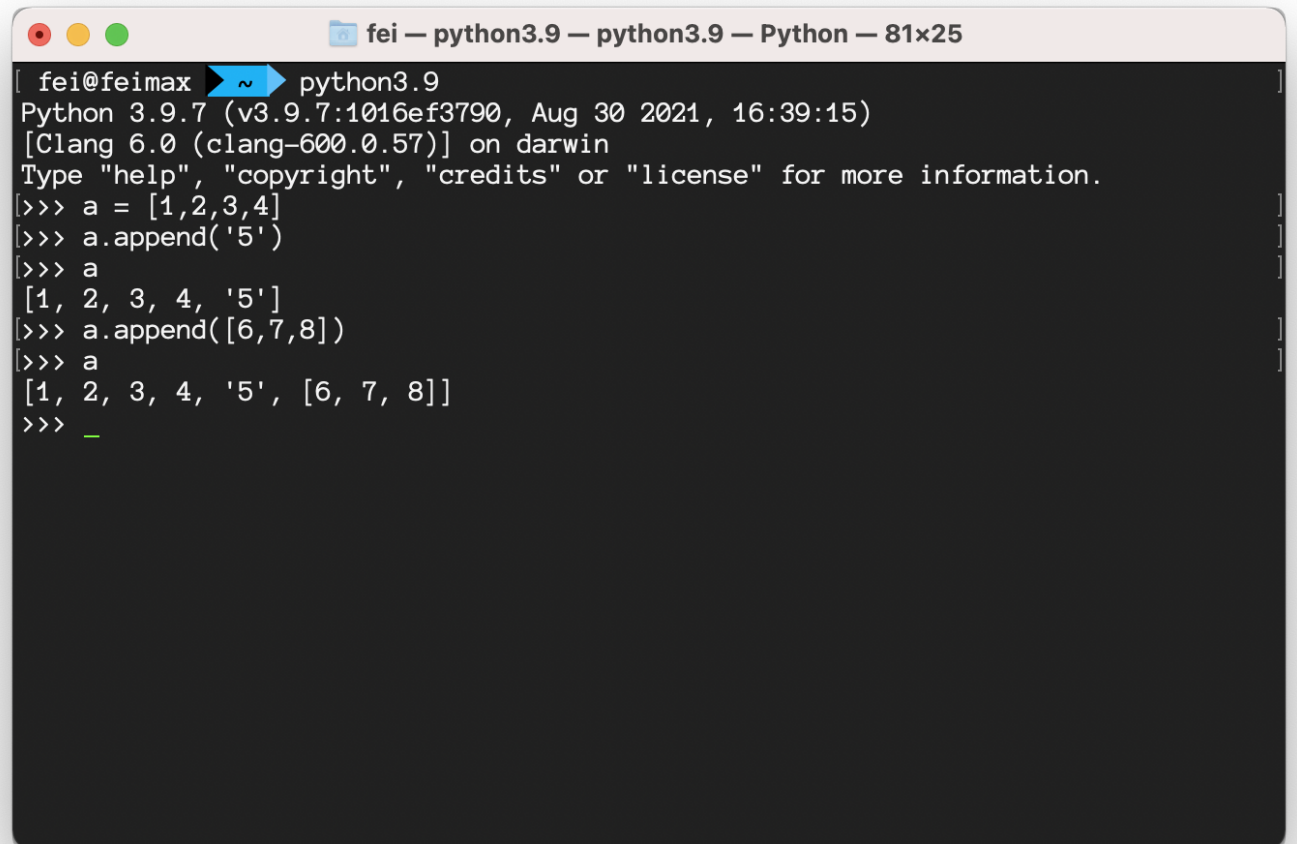
```
def TCP_processor(connection_socket, address): 1 usage
    global records
    print(address, ' connected')
```

Data structure

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

List

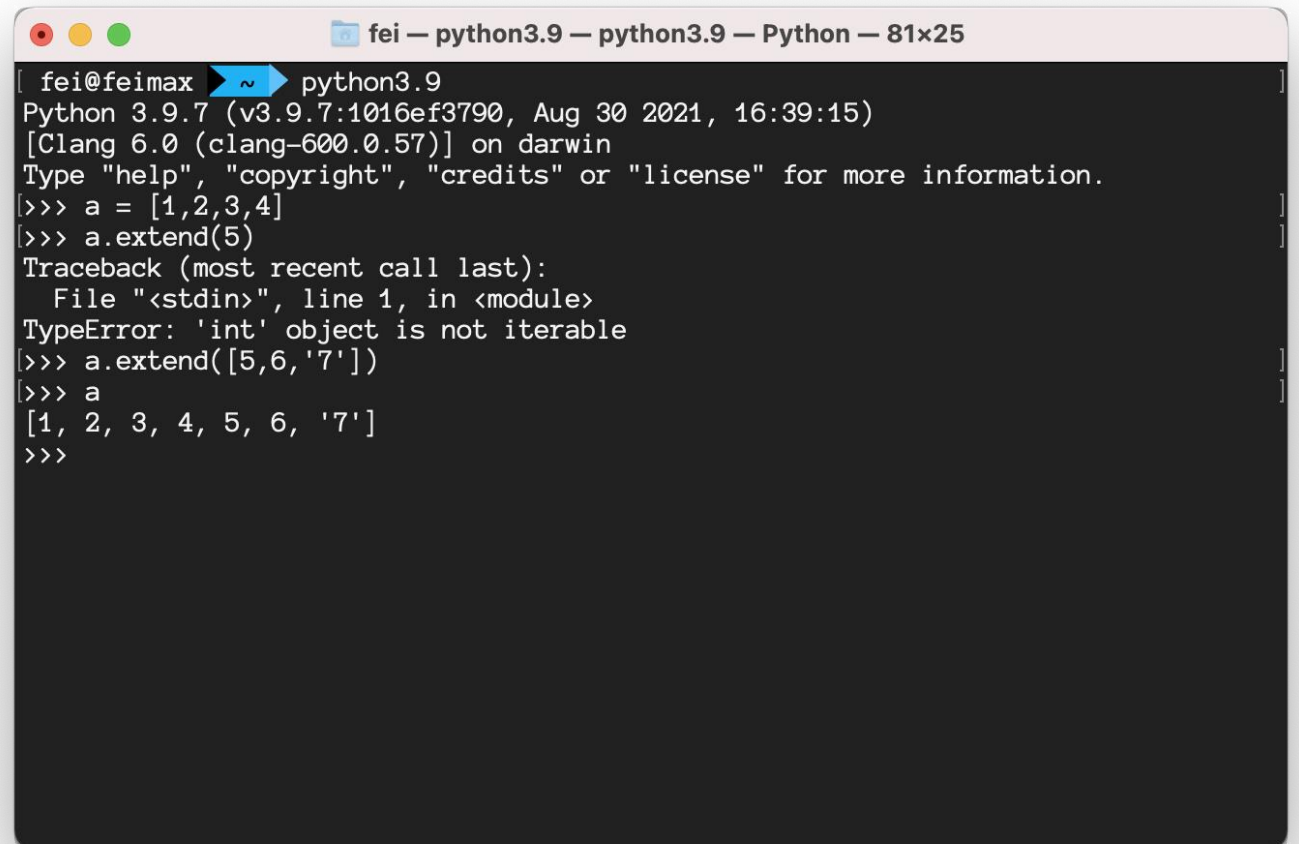
- Review
 - `list.append(x)`



```
fei — python3.9 — python3.9 — Python — 81x25
[ fei@feimax ~ ]$ python3.9
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [1,2,3,4]
>>> a.append('5')
>>> a
[1, 2, 3, 4, '5']
>>> a.append([6,7,8])
>>> a
[1, 2, 3, 4, '5', [6, 7, 8]]
>>> _
```

List

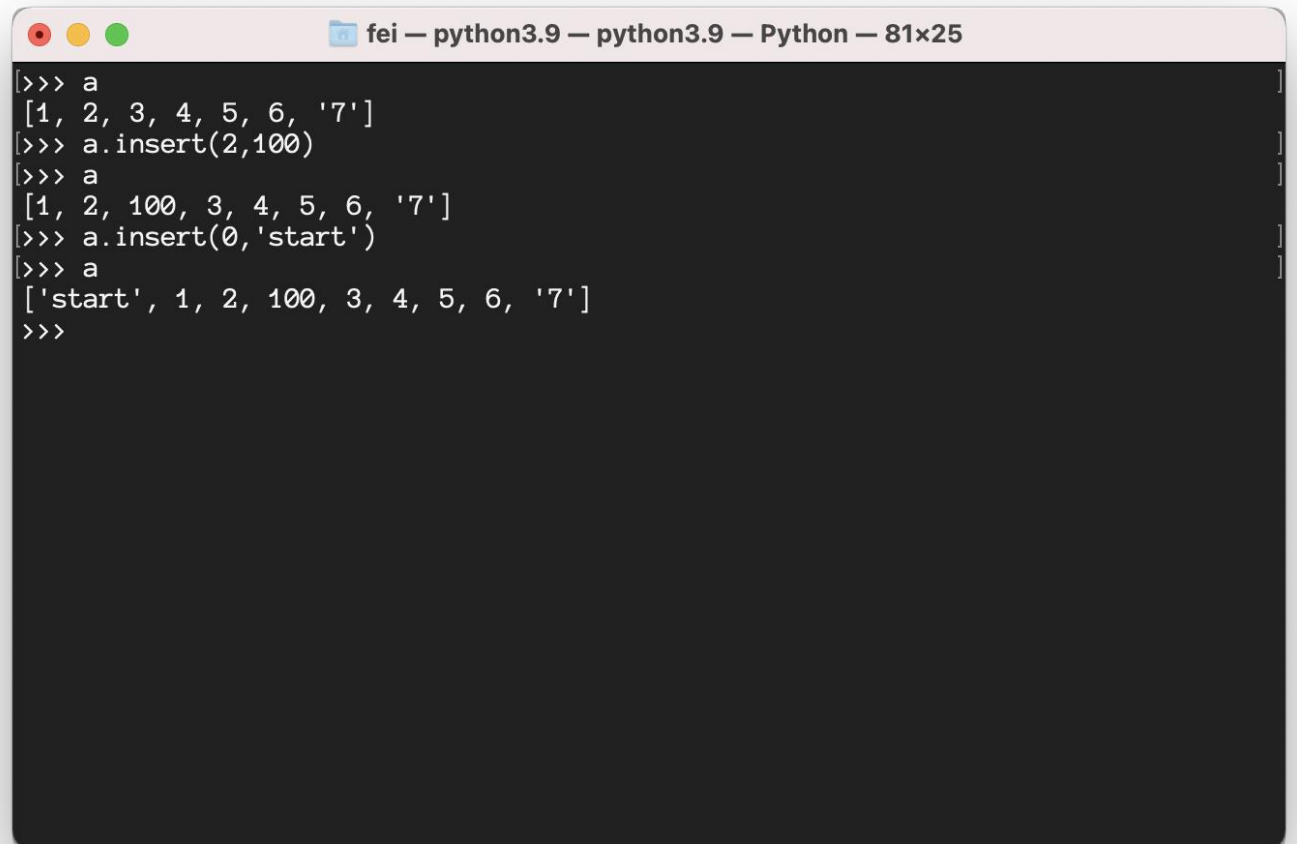
- Review
 - `list.append(x)`
 - `list.extend(iterable)`



```
fei — python3.9 — python3.9 — Python — 81x25
[ fei@feimax ~ ] python3.9
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [1,2,3,4]
>>> a.extend(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> a.extend([5,6,'7'])
>>> a
[1, 2, 3, 4, 5, 6, '7']
>>>
```

List

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

A screenshot of a Python terminal window. The window has a title bar with three colored buttons (red, yellow, green) on the left and a title text "fei — python3.9 — python3.9 — Python — 81x25" on the right. The terminal background is dark with light-colored text. The code being executed is as follows:

```
[>>> a
[1, 2, 3, 4, 5, 6, '7']
>>> a.insert(2,100)
>>> a
[1, 2, 100, 3, 4, 5, 6, '7']
>>> a.insert(0,'start')
>>> a
['start', 1, 2, 100, 3, 4, 5, 6, '7']
>>>
```

List

- Review

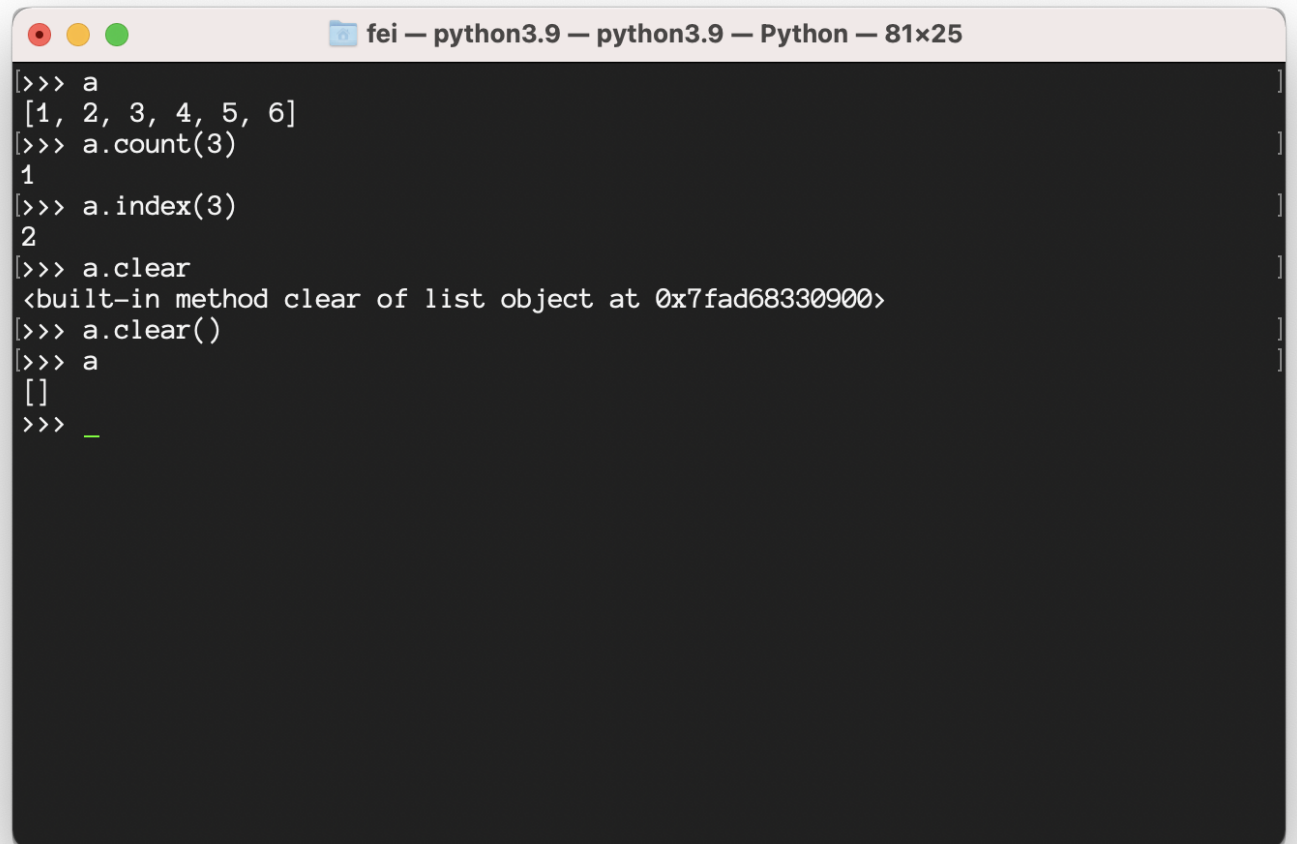
- `list.append(x)`
- `list.extend(iterable)`
- `list.insert(i, x)`
- `list.pop(i)`

```
fei — python3.9 — python3.9 — Python — 81x25
[>>> a
[1, 2, 3, 4, 5, 6, '7']
[>>> a.insert(2,100)
[>>> a
[1, 2, 100, 3, 4, 5, 6, '7']
[>>> a.insert(0,'start')
[>>> a
['start', 1, 2, 100, 3, 4, 5, 6, '7']
[>>> a.pop(3)
100
[>>> a
['start', 1, 2, 3, 4, 5, 6, '7']
[>>> a.pop()
'7'
[>>> a
['start', 1, 2, 3, 4, 5, 6]
[>>> a.pop(0)
'start'
[>>> a
[1, 2, 3, 4, 5, 6]
[>>> _
```

List

- Review

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

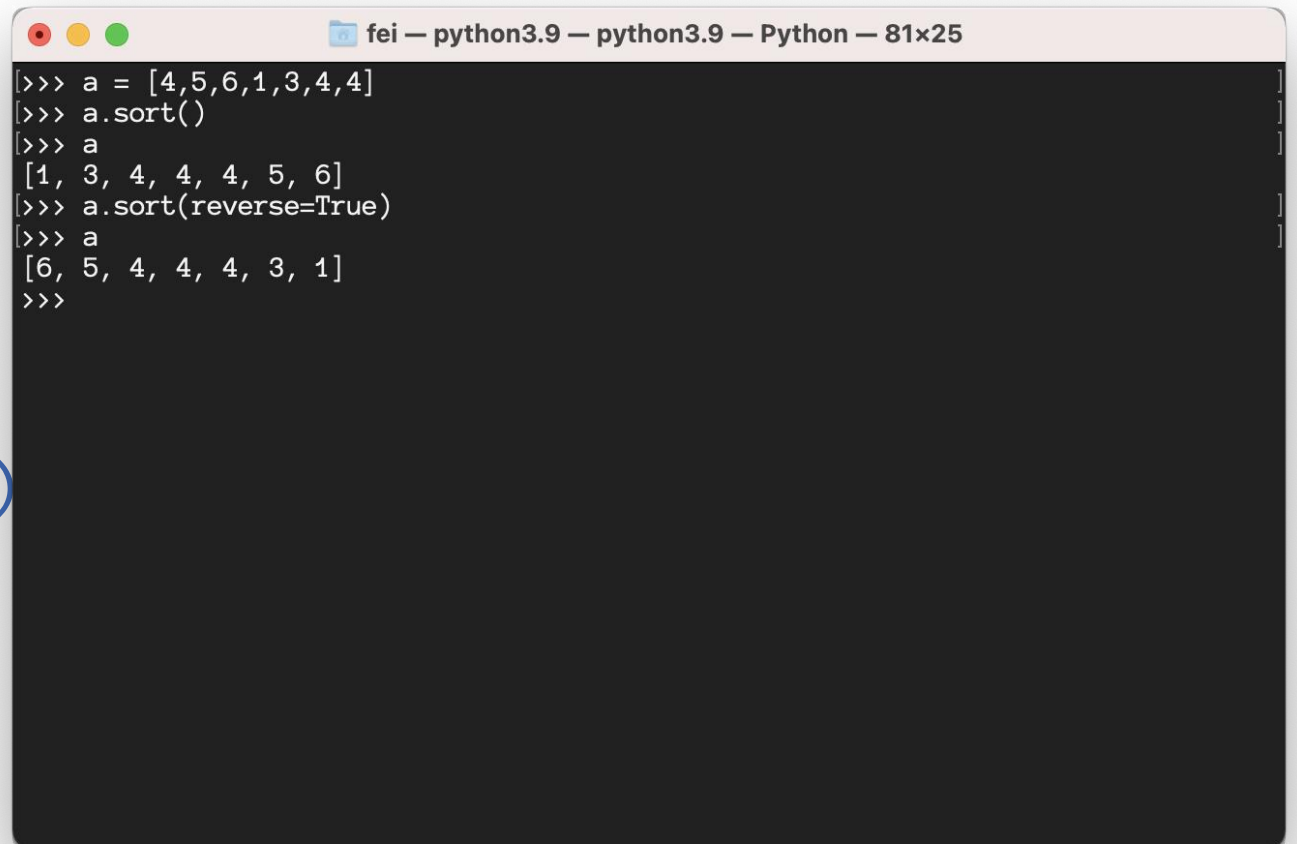
A screenshot of a Python terminal window titled "fei — python3.9 — python3.9 — Python — 81x25". The terminal shows a series of commands and their outputs: creating a list 'a' with values [1, 2, 3, 4, 5, 6], counting the number of '3's (output: 1), finding the index of '3' (output: 2), and then clearing the list (output: <built-in method clear of list object at 0x7fad68330900>). Finally, the list 'a' is shown as an empty list [].

```
>>> a
[1, 2, 3, 4, 5, 6]
>>> a.count(3)
1
>>> a.index(3)
2
>>> a.clear
<built-in method clear of list object at 0x7fad68330900>
>>> a.clear()
>>> a
[]
>>> _
```


List

- Review

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

A screenshot of a Python terminal window titled "fei — python3.9 — python3.9 — Python — 81x25". The terminal shows a series of commands and their outputs: first, a list 'a' is initialized with [4, 5, 6, 1, 3, 4, 4]; then 'a.sort()' is called, resulting in the list [1, 3, 4, 4, 4, 5, 6]; finally, 'a.sort(reverse=True)' is called, resulting in the list [6, 5, 4, 4, 4, 3, 1].

```
>>> a = [4,5,6,1,3,4,4]
>>> a.sort()
>>> a
[1, 3, 4, 4, 4, 5, 6]
>>> a.sort(reverse=True)
>>> a
[6, 5, 4, 4, 4, 3, 1]
>>>
```

List

- Review

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

```
fei — python3.9 — python3.9 — Python — 81x25
[>>> a
[6, 5, 4, 4, 4, 3, 1]
[>>> b = a
[>>> a[0] = 0
[>>> b
[0, 5, 4, 4, 4, 3, 1]
[>>> b = a.copy()
[>>> a[0]=100
[>>> b
[0, 5, 4, 4, 4, 3, 1]
[>>> a
[100, 5, 4, 4, 4, 3, 1]
[>>>
```

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)]
```

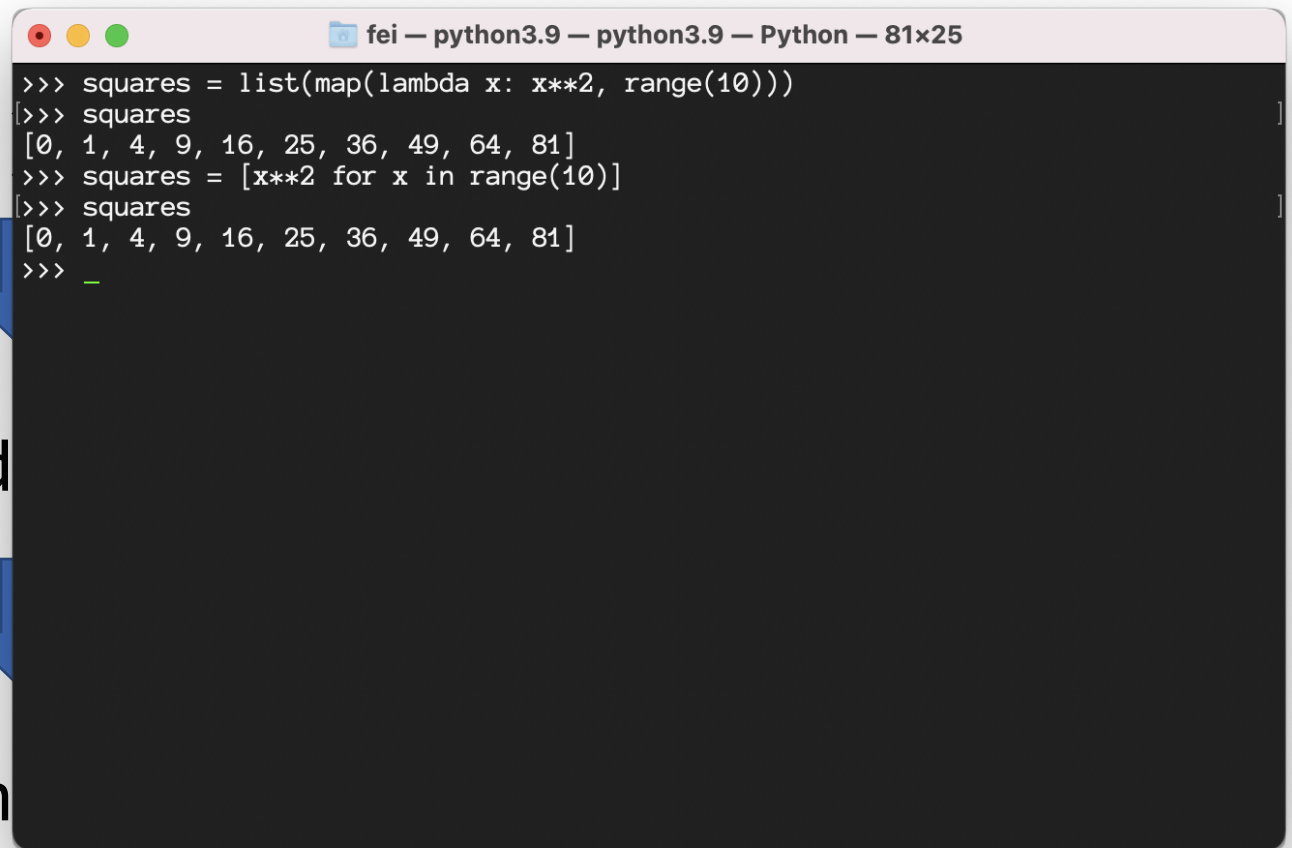
List Comprehensions

```
squares = []
```

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

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

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



```
fei — python3.9 — python3.9 — Python — 81x25  
>>> squares = list(map(lambda x: x**2, range(10)))  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
>>> squares = [x**2 for x in range(10)]  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
>>> _
```

Tuple

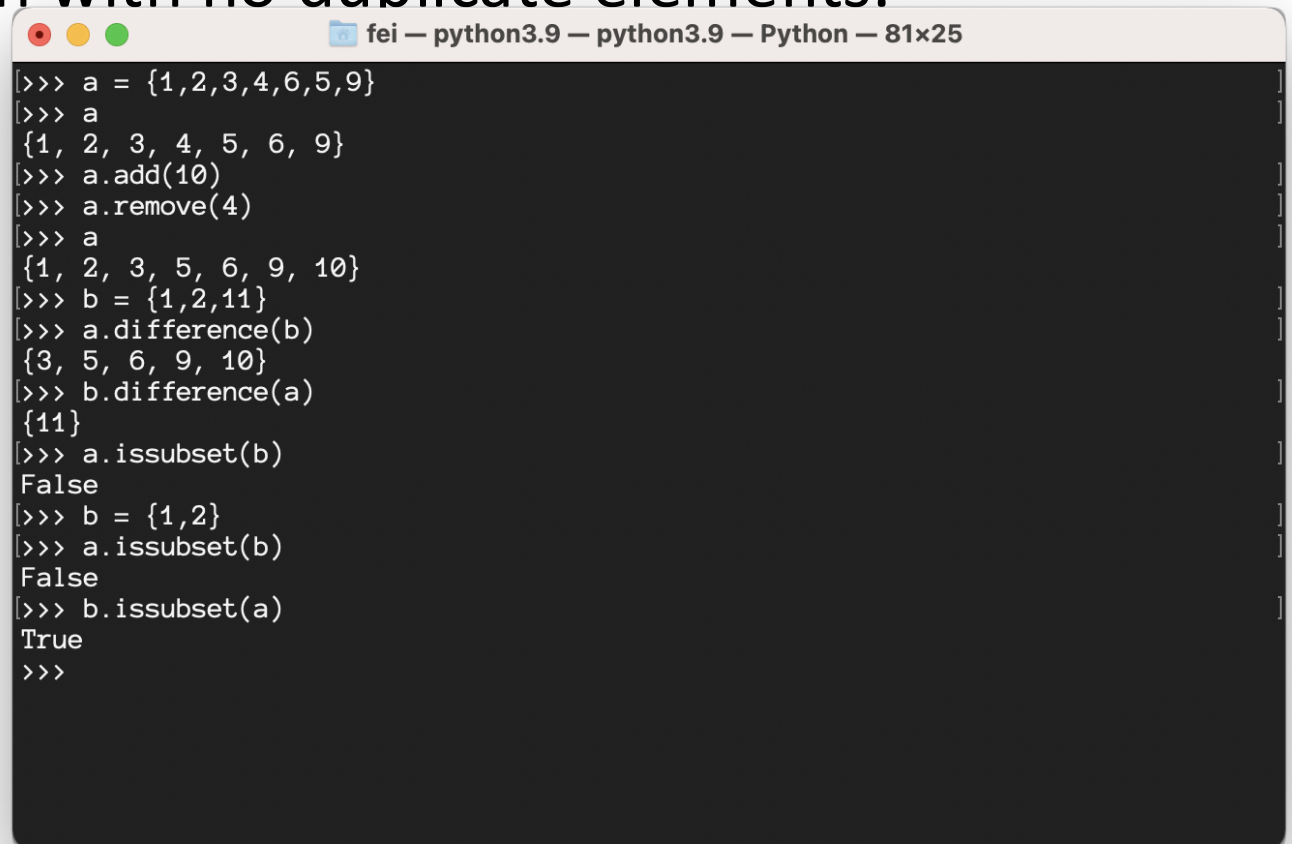
- Another sequence data type.
- `t = (1, 2, '4', 'string')`
- Sequence data types: indexing, slicing ... operations
 - list
 - tuple
 - range
- An immutable list

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)`

A screenshot of a Python terminal window titled "fei — python3.9 — python3.9 — Python — 81x25". The terminal shows a series of commands and their outputs for set operations. The commands include creating set 'a' with elements {1, 2, 3, 4, 6, 5, 9}, adding 10, removing 4, creating set 'b' with elements {1, 2, 11}, and using difference, issubset, and issuperset methods. The outputs show the resulting sets and boolean values for the subset checks.

```
>>> a = {1,2,3,4,6,5,9}
>>> a
{1, 2, 3, 4, 5, 6, 9}
>>> a.add(10)
>>> a.remove(4)
>>> a
{1, 2, 3, 5, 6, 9, 10}
>>> b = {1,2,11}
>>> a.difference(b)
{3, 5, 6, 9, 10}
>>> b.difference(a)
{11}
>>> a.issubset(b)
False
>>> b = {1,2}
>>> a.issubset(b)
False
>>> b.issubset(a)
True
>>>
```

Dictionary

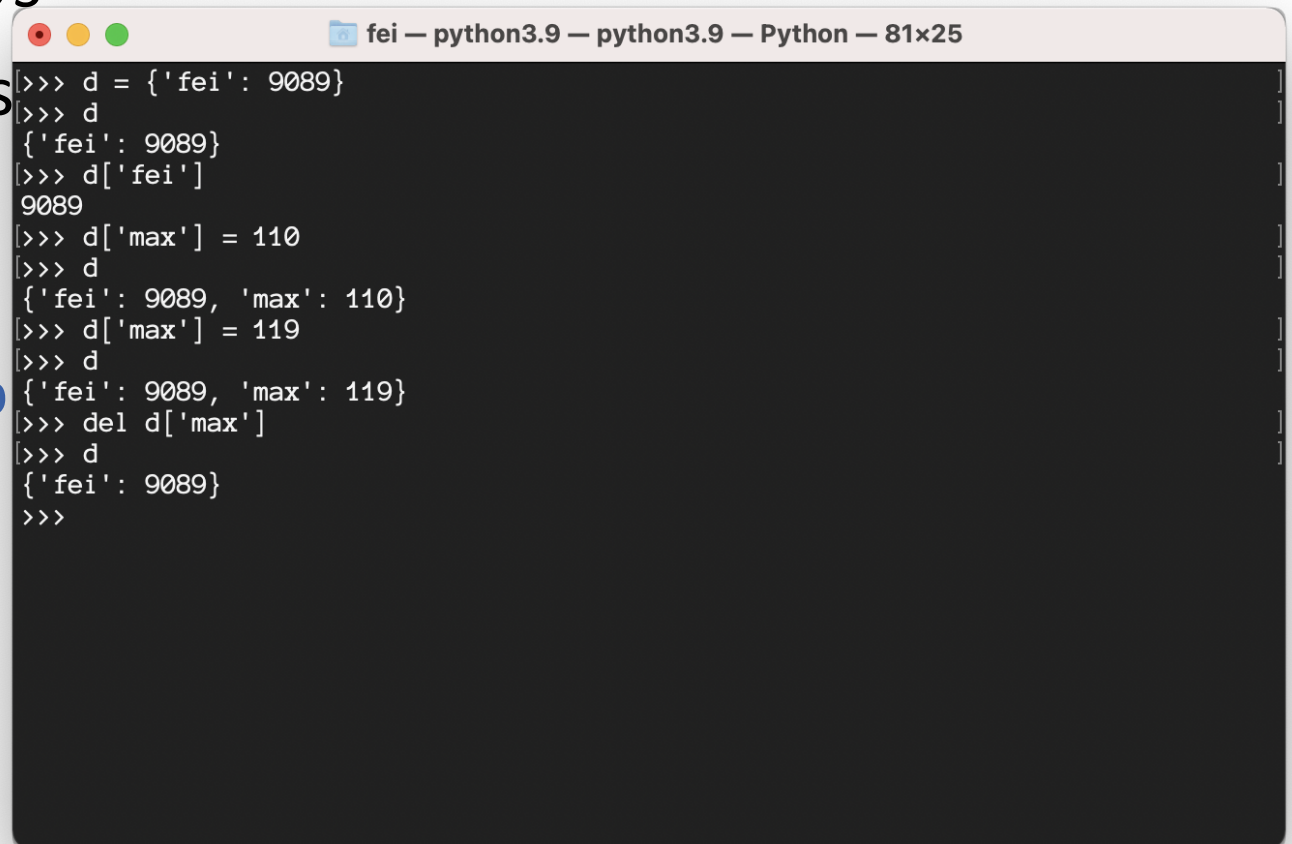
- Dictionaries are indexed by *keys*
- Keys can be numbers or strings
- key - value pairs
- First example:

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


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
```



```
fei — python3.9 — python3.9 — Python — 81x25  
>>> d = {'fei': 9089}  
>>> d  
{'fei': 9089}  
>>> d['fei']  
9089  
>>> d['max'] = 110  
>>> d  
{'fei': 9089, 'max': 110}  
>>> d['max'] = 119  
>>> d  
{'fei': 9089, 'max': 119}  
>>> del d['max']  
>>> d  
{'fei': 9089}  
>>>
```

IMPORTANT

Dictionary

- How to know the key exists or not?

```
fei — python3.9 — python3.9 — Python — 81x25
[>>> d
{'fei': 9089}
[>>> 'fei' in d
True
[>>> 'fei' in d.keys()
True
[>>> d.keys()
dict_keys(['fei'])
[>>> list(d.keys())
['fei']
[>>> 'max' in d.keys()
False
[>>> 9089 in d
False
[>>> _
```

IMPORTANT

Dictionary

- How to enumerate the dict?

A screenshot of a Python terminal window. The title bar at the top reads "fei — python3.9 — python3.9 — Python — 81x25". The terminal has a dark background with white text. It shows a dictionary 'd' being created with two items: 'fei' with value 9089 and 'max' with value 'foobar'. Then, a loop uses 'd.items()' to iterate over the dictionary, printing each key-value pair on a new line.

```
[>>> d
{'fei': 9089, 'max': 'foobar'}
[>>> for k,v in d.items():
...     print(k, v)
...
fei 9089
max foobar
>>>
```

IMPORTANT

Dictionary

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

JSON

IMPORTANT

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

IMPORTANT

JSON

- JavaScript Object Notation
- dict <-> JSON

```
fei — python3.9 — python3.9 — Python — 81x25
[>>> import json
[>>> d
{'fei': 9089, 'max': 'foobar', 'a': 1}
[>>> json.dumps(d)
'{"fei": 9089, "max": "foobar", "a": 1}'
[>>> json.loads('{"fei": 9089, "max": "foobar", "a": 1}')
{'fei': 9089, 'max': 'foobar', 'a': 1}
[>>> json_str = json.dumps(d)
[>>> type(json_str)
<class 'str'>
[>>> dict_from_json = json.loads('{"fei": 9089, "max": "foobar", "a": 1}')
[>>> type(dict_from_json)
<class 'dict'>
[>>>
```

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, 678132648176981237, 1.414, 'Hello'] using socket

4 bytes 8 bytes 4 bytes x bytes
unsigned int unsigned long long float string

info = [1, 678132648176981237, 1.414, 'Hello']

binary_data = struct.pack('!IQf', info[0], info[1], info[2]) + info[3].encode()

print(binary_data)

Use + to link two byte streams to one

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)

File operation

“***open***” method is exploit to operate files:

- `f = open(file='filename.xxx', mode='r')`
 - “open” method open a file and return a stream (a file object)
 - `file` is a path of filename:
 - if the file is in the **current working directory**, relative path: eg. record.txt, or folder_a/record.txt
 - Otherwise, the **full path (absolute path)** : eg. (Win)D:\your_dir\record.txt or (Mac)/Users/your_home/record.txt
 - `mode` specifies the mode in which the file is opened.

current working directory: default is the folder that has the current executing .py file. You can also change the CWD.

File open modes

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode, used for media files
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newline mode (deprecated)

* We used the red modes normally

Text file write

- The following methods can be used:
 - `f.write(Str)` # Write a str to a file
 - `f.writelines(StrList)` # Write a list of str to a file

```
cont = ['Hello, Li Lei.\n', 'Hello, Han Meimei.\n',  
        'How are you?\n',  
        'Fine, thank you. And you?\n',  
        'I am fine too.\n']  
f = open('/Users/fei/lesson1.txt', 'w')  
# Write all the list once  
f.writelines(cont)  
# Write line by line  
for l in cont:  
    f.write(l)  
  
f.close()
```

Text file read

- The following methods can be used:
 - `f.read(n)` # read n chars(including `\n`), `n=-1` means read all
 - `f.readline()` # read one line (including `\n`)
 - `f.readlines()` # return a list of all lines

```
f = open('/Users/fei/lesson1.txt', 'r')
print(f.read())    # Read all contents

f.seek(0)  # Set the file pointer to the beginning
print(f.read(10)) # Read 10 chars

f.close()
```

```
# Use readline
f = open('/Users/fei/lesson1.txt', 'r')

while True:
    line = f.readline()    # Read one line
    if line: # if the line is not empty, print, otherwise break
        print(line)
    else:
        break

f.close()

# Use readlines
f = open('/Users/fei/lesson1.txt', 'r')

cont = f.readlines()
# Iterate all the lines
for line in cont:
    print(line)

f.close()
```

Text file appending

- Open a file with 'a' mode. Then, write something. The new content will append to the end of that file. If there is no that file, it will create a new one. It is very useful to record logs.

```
f = open('/Users/fei/lesson1.txt', 'a')  
  
f.write('Bye.\n')  
  
f.close()
```


Binary file read

- For binary files, still we have those 3 methods to read them:
 - `f.read(n)` # read n bytes(including `\n`), `n=-1` means read all
 - `f.readline()` # read one line (including `\n`)
 - `f.readlines()` # return a list of all lines

```
# Open with text read mode
f = open('/Users/fei/lesson1.txt', 'r')
print(type(f.read()))
f.close()

# Open with binary read mode, what's the difference?
f = open('/Users/fei/lesson1.txt', 'rb')
print(type(f.read()))
f.close()
```

Binary file write

- The following methods can be used:
 - `f.write(bytes)` # Write some bytes to a file
- Binary file operation is very powerful, but we don't go deep today.
 - We just open a binary file (jpeg image), read all the data, and then write the same data to another jpg file.

```
f = open('/Users/fei/a.jpg', 'rb')
# Read all the binary bytes from a.jpg
image_data = f.read()

f.close()

# Write to a new jpg file
f = open('/Users/fei/aa.jpg', 'wb')
f.write(image_data)
f.close()
```

File operation demo

<https://box.xjtlu.edu.cn/f/20e8ff4ad71644e1ba29/>



Lab practice

- Design a pair of UDP based server and client programs:
 - Client sends a small jpeg image, named xjtlu.jpg to server side;
 - Server receives the image data, and save it as xjtlu1.jpg on server;
 - Server sends back a small jpeg image, named xjtlu1.jpg to client;
 - Client receives the image data, and save it as xjtlu2.jpg on client.
- Design a pair of TCP based server and client program, the requirements are the same as above.
- xjtlu.jpg: <https://box.xjtlu.edu.cn/f/2dfde6edfef14cffa5eb/?dl=1>
- No special module(lib) is need. (opencv is not needed!)
- Your codes should be executed on different machines (one real PC and one virtual machine, or two virtual machines).
- No submission, no checking!

In class test 1 (5 points)

- A multiplayer “**number guessing**” game using sockets programming.
- Should have one server to manages the game logic .
- Should have some (at least two) players join the game.
- Detailed rules will be released during each lab session.
- The requirements are slightly different for different lab sessions
 - Pls read the requirement carefully.
- You can prepare for the in class test 1.
- You will lose 1 points if you cannot submit during the lab time.
- You will lose 2 points if you cannot submit within 24 hours.

Thanks