Logistics

- No class next Monday (10/10)
- Homework will be out mid this week
- Thu office hour is not working
  - New hours: Mon 1pm – 2pm (in person/virtual), Thu 12noon – 1pm (virtual)

Lecture 4. Python Primer – Part 4

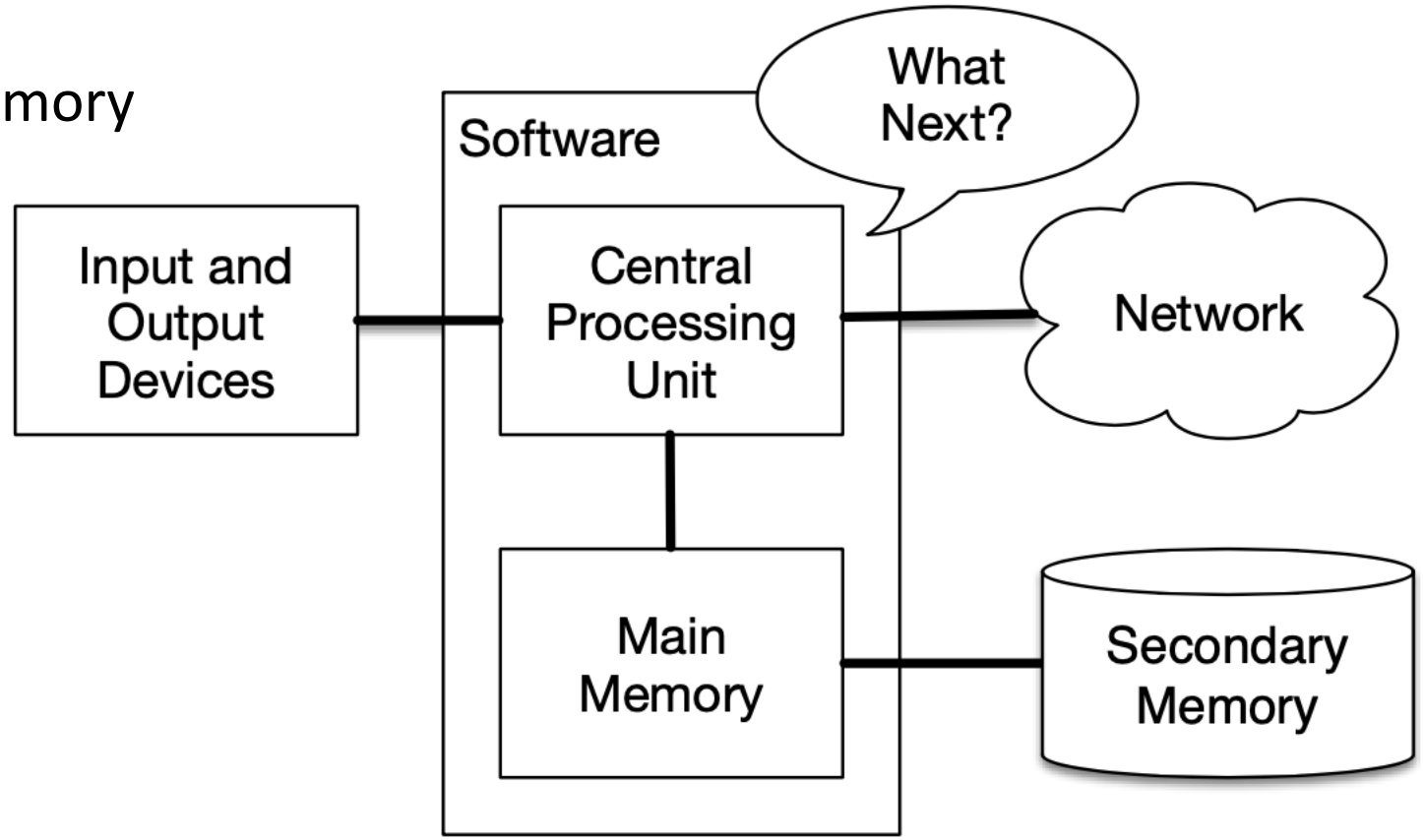
Chao Chen

Stony Brook University

Oct. 03, 2022

Files

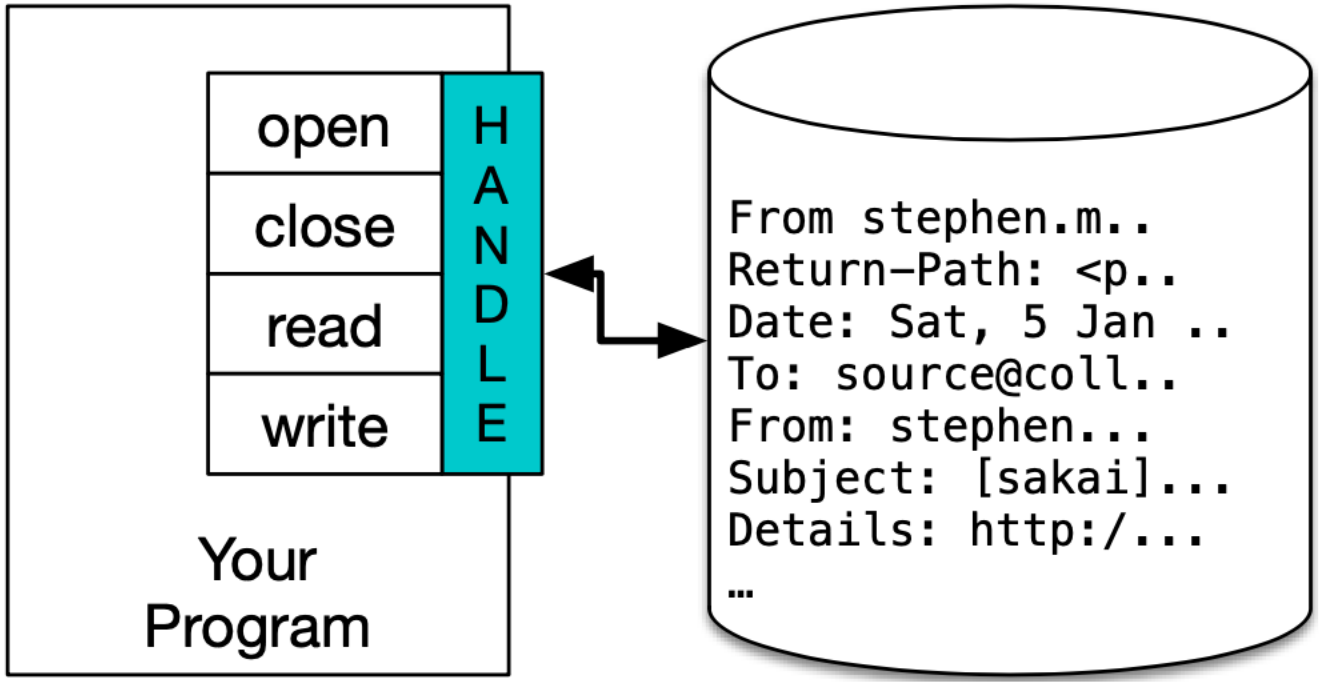
- Files – stored in secondary memory
  - Read into main memory
  - Write into secondary memory



File

- file handles – returned by a successful ”open” operation

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='cp1252'>
```



File

- If open failed (file does not exist, typo in name, etc) an error will occur

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'stuff.txt'
```

- Use try/except to avoid the error. (How?)

End-of-line

- End-of-line – a special character to separate lines

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

File

- A file can be read line by line (automatically split by '\n')

```
fhand = open('mbox-short.txt')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)
```

Read a single line into the main memory

- Or be read in as a single string

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

Read the whole file into the main memory

Convenient –
but only do this if the file can fit the memory.

File -- scanning

- Scanning through lines and only use the ones starting with “From:”

```
fhand = open('mbox-short.txt')
count = 0
for line in fhand:
    if line.startswith('From:'):
        print(line)
```

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu
...
```

- Note: The extra empty line is due to `\n` by the end of each line.
- Use rstrip

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
...
```

File – scanning through a file

- Another way to code: skipping lines which do not start with “From:”
- Same function, but conceptually this is important
  - We are skipping lines that does not satisfy certain criterion

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    # Skip 'uninteresting lines'
    if not line.startswith('From:'):
        continue
    # Process our 'interesting' line
    print(line)

# Code: http://www.py4e.com/code3/sear
```

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1: continue
    print(line)
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac
From: david.horwitz@uct.ac.za
```

File – scanning lines

- Another example what does this do?

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1: continue
    print(line)

# Code: http://www.py4e.com/code3/search4.py
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
Author: david.horwitz@uct.ac.za
...
```

File -- pointer and seek

- What if I want to scan twice or more?
- File reading follows a pointer
- After scanning once, you need to reset the pointer to the beginning

```
# %load sample2.txt
aaa
bbb
ccc
ddd

```

```
f = open('sample2.txt')
print('Reading once')
count = 0
for line in f:
    count = count + 1
    if line.startswith('a'):
        print(line.rstrip())
print('Read ', count, ' lines.\n')

count = 0
print('Reading twice')
for line in f:
    count = count + 1
    if line.startswith('b'):
        print(line.rstrip())

print('Read ', count, ' lines.')
```

```
Reading once
aaa
Read  4  lines.

Reading twice
Read  0  lines.
```

File -- pointer and seek

- seek(offset) - set pointer to …
  - Examples: f.seek(0), f.seek(5)
  - Note some doc says seek(offset,whence) - this is not supported anymore

```
# %load sample2.txt
aaa
bbb
ccc
ddd

```

```
f = open('sample2.txt')
print('Reading once')
count = 0
for line in f:
    count = count + 1
    if line.startswith('a'):
        print(line.rstrip())
print('Read ', count, ' lines.\n')

f.seek(0)

count = 0
print('Reading twice')
for line in f:
    count = count + 1
    if line.startswith('b'):
        print(line.rstrip())

print('Read ', count, ' lines.')
```

```
Reading once
aaa
Read  4  lines.

Reading twice
bbb
Read  4  lines.
```

File - other useful reading functions

- tell() – find out current position
  - f.seek(f.tell() + 3) – move forward for 3 chars
- lines = f.readlines() # read in all lines as a list of strings
  # (one line per item)
- line = f.readline() # read one line into a string
- https://docs.python.org/3/tutorial/inputoutput.html#methods-of-file-objects
- If the file is not text file (in binary format)
  - Open('sample', 'rb')
  - read will read in a byte each time.
- https://docs.python.org/3/library/functions.html?highlight=open#open

File – ask the user for file name

- Get the file name from users

```
fname = input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)

# Code: http://www.py4e.com/code3/search6.py
```

```
python search6.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python search6.py
Enter the file name: mbox-short.txt
There were 27 subject lines in mbox-short.txt
```

Million-dollar question for a programmer:

What could possibly go wrong?

File – user input his/her own file name

What if our user types something that is not a file name?

```
python search6.py
Enter the file name: missing.txt
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'missing.txt'

python search6.py
Enter the file name: na na boo boo
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'na na boo boo'
```

How to handle this automatically without causing an error?

File – avoid errors by try/except

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)

# Code: http://www.py4e.com/code3/search7.py
```

```
python search7.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python search7.py
Enter the file name: na na boo boo
File cannot be opened: na na boo boo
```

File – writing a file

To write a file, you have to open it with mode “w” as a second parameter:

```
>>> fout = open('output.txt', 'w')
>>> print(fout)
<_io.TextIOWrapper name='output.txt' mode='w' encoding='cp1252'>
```

File - write

- Write function, return how many characters have been written

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
24

>>> line2 = 'the emblem of our land.\n'
>>> fout.write(line2)
24
```

File – close when you finish writing

- Remember to close the file when you are done (especially when writing)

```
>>> fout.close()
```

- Without closing properly, file may be unreadable
- A good read: https://realpython.com/why-close-file-python/

File – can also use print

- In print, use file = ??? to redirect output to a file that has been opened

```
1  f = open('sample2.txt','a')
2  print('www',file=f)
3  f.close()
```

```
1  # %load 'sample2.txt'
2  aaa
3  bbb
4  ccc
5  ddd
6  www
7
```

File – other useful functions

writelines(*lines*, /)

Write a list of lines to the stream. Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

truncate(*size=None*, /)

Resize the stream to the given *size* in bytes (or the current position if *size* is not specified).

readable()

Return `True` if the stream can be read from. If `False`, `read()` will raise `OSError`.

seekable()

Return `True` if the stream supports random access. If `False`, `seek()`, `tell()` and `truncate()` will raise `OSError`.

writable() ¶

Return `True` if the stream supports writing. If `False`, `write()` and `truncate()` will raise `OSError`.

Useful link: https://docs.python.org/3/library/io.html

File – more about the open modes

```
>>> fout = open('output.txt', 'w')
```

- ‘r’ – read
  - If the file does not exist, error
- ‘w’ open
  - If the file does not exist, create
  - If the file exists, truncate – remove whatever is in the file
- ‘a’ – if the file exists, append to the end instead of removing
- ‘+’ – allow both reading and writing at the same time

File – more about the open modes

- Check out: https://mkyong.com/python/python-difference-between-r-w-and-a-in-open/

	r	r+	w	w+	a	a+
read	*	*		*		*
write		*	*	*	*	*
create			*	*	*	*
truncate			*	*		
position at start	*	*	*	*		
position at end					*	*

Debugging

- repr(s) – string representation

```
>>> s = '1 2\t 3\n 4'
>>> print(s)
1 2  3
 4
```

```
>>> print(repr(s))
'1 2\t 3\n 4'
```

Exercise 1: Write a program to read through a file and print the contents of the file (line by line) all in upper case. Executing the program will look as follows:

```
python shout.py
Enter a file name: mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN  5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
     BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
     SAT, 05 JAN 2008 09:14:16 -0500
```

You can download the file from

www.py4e.com/code3/mbox-short.txt

Exercise 2: Write a program to prompt for a file name, and then read through the file and look for lines of the form:

```
X-DSPAM-Confidence:0.8475
```

When you encounter a line that starts with "X-DSPAM-Confidence:" pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence.

```
Enter the file name: mbox.txt
Average spam confidence: 0.894128046745

Enter the file name: mbox-short.txt
Average spam confidence: 0.750718518519
```

Test your file on the `mbox.txt` and `mbox-short.txt` files.

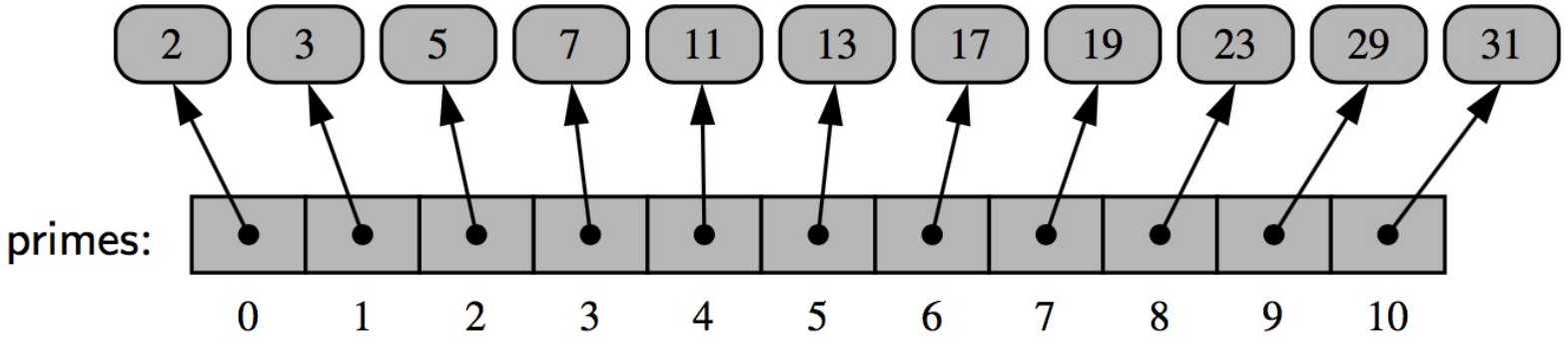
Exercise 3: Sometimes when programmers get bored or want to have a bit of fun, they add a harmless *Easter Egg* to their program Modify the program that prompts the user for the file name so that it prints a funny message when the user types in the exact file name "na na boo boo". The program should behave normally for all other files which exist and don't exist. Here is a sample execution of the program:

```
python egg.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python egg.py
Enter the file name: missing.tyxt
File cannot be opened: missing.tyxt

python egg.py
Enter the file name: na na boo boo
NA NA BOO BOO TO YOU - You have been punk'd!
```

We are not encouraging you to put Easter Eggs in your programs; this is just an exercise.

List

- Sequence of elements with index



List construction

- Create with [elem1, elem2, elem3]

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print(cheeses, numbers, empty)
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

- Can be mixed types – elements have different types
- Can be nested (a list as an element)

```
['spam', 2.0, 5, [10, 20]]
```

List - indexing

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
```

- Index – x[idx]

```
>>> print(cheeses[0])
Cheddar
```

- Mutable – can change elements

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print(numbers)
[17, 5]
```

List – in operator

The `in` operator also works on lists.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

List – travers through a list

- Traversing using for loop

```
for cheese in cheeses:
    print(cheese)
```

- Can also enumerate through all possible indices (can be more flexible)

```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

range(n) -- equivalent to the list [0, 1, 2, …, n-1]
but saved implicitly

List – empty list, length of a list

- An empty list has zero length (no element)

```
for x in empty:
    print('This never happens.')
```

- Question: what is the length of the following list?

```
['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
```

Answer: 4 – a list is still counted as one item

List - operators

The + operator concatenates lists:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

- Multiplication “*” repeat a list for multiple times

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List -- slices

The slice operator also works on lists:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']

>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

List - slice

- Using slice to update multiple elements

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print(t)
['a', 'x', 'y', 'd', 'e', 'f']
```

- Try t[first:last:step]

```
1 t = ['a','b','c','d','e','f']
2 t[1:6:2] = ['x','y','z']
3 print(t)
```

```
['a', 'x', 'c', 'y', 'e', 'z']
```

List - functions

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print(t)
['a', 'b', 'c', 'd']
```

`extend` takes a list as an argument and appends all of the elements:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print(t1)
['a', 'b', 'c', 'd', 'e']
```

List - functions

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print(t)
['a', 'c']
>>> print(x)
b
```

If you don't need the removed value, you can use the **del** operator:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print(t)
['a', 'c']
```

List - functions

- Knowing the value but not index – use remove

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print(t)
['a', 'c']
```

The return value from `remove` is `None`.

```
1  a = [1,2,3,3,2,1]
2  a.remove(2)
3  print(a)
```

```
[1, 3, 3, 2, 1]
```

- Note: only remove the first occurrence
- What if the value does not belong to the list?

List - remove

To remove more than one element, you can use **del** with a slice index:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> print(t)
['a', 'f']
```

As usual, the slice selects all the elements up to, but not including, the second index.

List – functions operating on list

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25
```

The `sum()` function only works when the list elements are numbers. The other functions (`max()`, `len()`, etc.) work with lists of strings and other types that can be comparable.

The end