Files

Lecture 05.01

By Marina Barsky

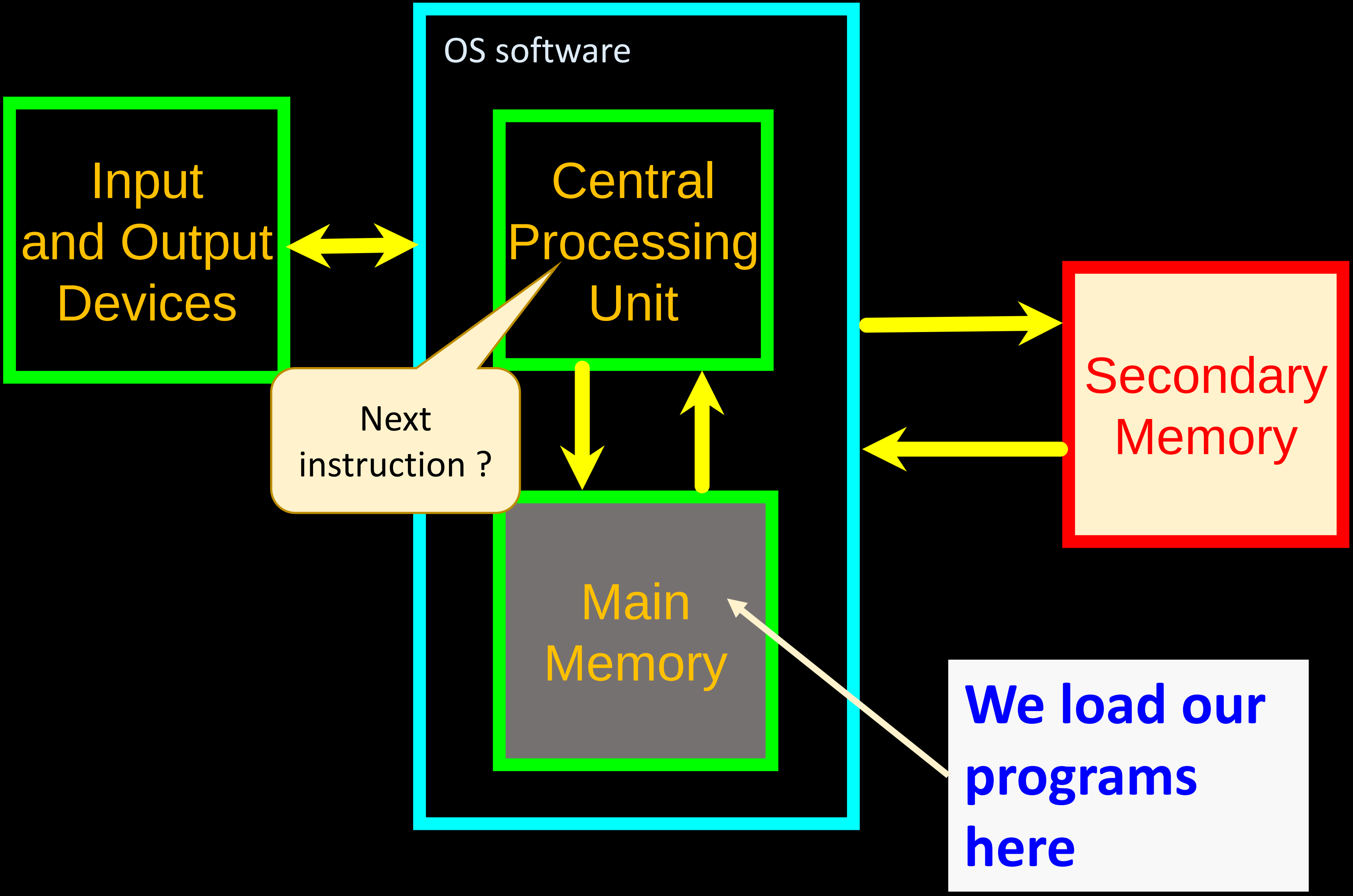
Sample data files:
emails.txt
results.txt
scores.txt
spam.txt

We live in time of “Big Data”

- We aggressively acquire and keep data forever
- We feel real freedom when all data is available
- Implications for our live are enormous
- We use data for purposes different than it was primarily collected

- Question: How do we keep data forever?

Stored-program computer



How to store inputs and outputs permanently?

Magnetic disks

Solid State Disks

https://youtu.be/TvkIi6NVnqY

Simplify access to data on disk with Python

- We operate on files using an abstraction of type *File*
- File abstraction in Python knows how to:
    - Open a file (establish connection to a file on disk)
    - Read the entire file into a string
    - Read the file line-by-line
    - Close the file (disconnect from file on disk)

    - Create a new file
    - Write to a file

Opening a file

- Before we can read the contents of the file from disk into main memory, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the *open()* function
- open() returns a "file handle" - a variable used to perform operations on the file
- Similar to "File -> Open" in a Word Processor

Opening the file: syntax

```
handle = open(filename, mode)
```

- returns a **handle** to manipulate the file
- **filename** is a string
- **mode** is optional and should be 'r' if we are planning to read the file and '**w**' if we are going to write to the file (‘r’ is default)

On disk: spam.txt

```
I like poptarts and 42 and spam.
Will I get spam and poptarts for holidays?
I like spam poptarts
```

Reading from a file

```
>>> f = open( 'spam.txt', 'r' )
```

opens the file for reading and put handle to the file into variable f

```
>>> text = f.read()
```

reads the **whole** file and puts its content into string variable text

alternative...

```
>>> text = f.read(50)
```

reads the next 50 bytes from the file

```
>>> f.close()
```

closes the file (optional, but leaving files open can lead to problems...)

```
>>> text
'I like poptarts and 42 and spam.\nWill I ...
```

all the text (as one big string)

```
>>> wordList = text.split()
[ 'I', 'like', 'poptarts', ... ]
```

text.split() returns a list of each "word"

Writing to file

```
handle = open("test.txt", "w")
```

opens the file for writing and calls it `handle`

```
text = "Hello, world!"
handle.write(text)
```

Writes string *text* to `file`

```
handle.close()
```

Working with raw text files

Reading entire file content into a string

Counting letters

```
handle = open("test.txt", "w")
text = "Hello\nworld"
handle.write(text)
handle.close()

handle = open("test.txt")
content = handle.read()
print(content)
print("Total letters:", len(content))
handle.close()
```

Opening an **existing** file for writing (even if you never write anything), **erases all existing** content!

The newline character

- We use a special character called the "*newline*" to indicate when a line ends
- We represent it as \n in strings
- Newline is counted as **one character** - not two
- When passed to *print* function – prints on different lines

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

File operations are risky

- If file we try to access **is not there** or we cannot write to the specified file: program blows off
- Always surround file opening code with **try-except**

```
success = True
try:
    f = open('spam.txt','r')
except IOError as e:
    print ("Error opening file: ", e)
    success = False

if success:
    text = f.read()
    print (text)
```

Working with text files: Splitting text

- The *split()* method breaks a string into a list of words.
- By default, any number of whitespace characters is considered a word boundary.

```
        song = "The rain in Spain..."
List →  wds = song.split()
        print(wds)        ['The', 'rain', 'in', 'Spain...']
```

- If you want to use another string as a splitting boundary, you specify additional parameter to split:

```
print('red; blue; green'.split(';'))

          ['red', ' blue', ' green']
```

Opposite to split: join

- The inverse of the split method is *join()*.
- You choose a desired separator string, (often called the glue) and join the **list of strings** with the glue between each of the elements.

```
wds = ["red", "blue", "green"]
glue = ';'
s = glue.join(wds)
print(s)                              red;blue;green

print("***".join(wds))                red***blue***green
print("".join(wds))                   redbluegreen
```

Sanitizing text – pure words

```
song = "The rain in Spain..."

#replace all non-letter characters with spaces
s = ''
for x in song:
      if x.isalpha()
            s += x
      else
            s += ' '

t=s.split()
#splits across all spaces
print(t)
```

```
The rain in Spain
['The', 'rain', 'in', 'Spain']
```

Or with list comprehensions – pure words

```
song = "The rain in Spain..."

#replace all non-letter characters with spaces
t = [x if x.isalpha() else ' ' for x in song]

# but now t is a list of characters
#convert from list of letters back to string
clean_str =''.join(t)

print(clean_str.split())
```

```
The rain in Spain
['The', 'rain', 'in', 'Spain']
```

Counting words and letters

```
song = "The rain in Spain..."

#replace all non-letter characters with spaces
s = ''
for x in song:
      if x.isalpha()
            s += x
      else
            s += ' '

t=s.split()
#splits across all spaces
print(t)
```

How can we find out how many total proper words?

And how can we find out how many total English letters used?

```
The rain in Spain
['The', 'rain', 'in', 'Spain']
```

Counting number of words in a text file

```
def wc( filename ):
    """ word-counting program """
    f = open( filename, "r")
    text = f.read()
    f.close()

    words = text.split()
    print("There are", len(words), "words.")
```

See previous slide for how to produce only alpha - words

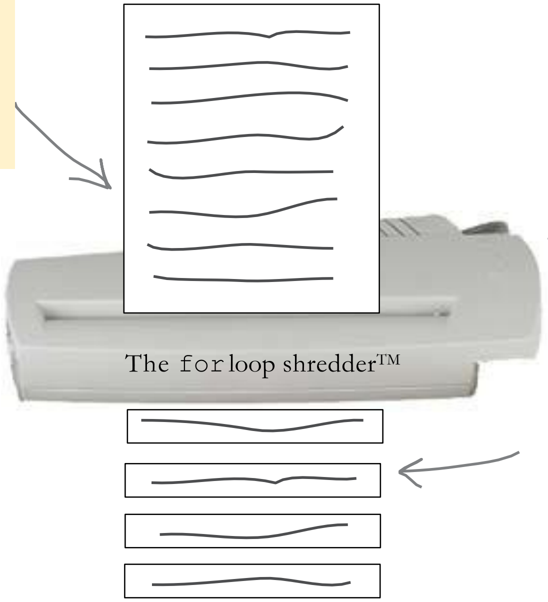
Working with structured data

Reading and parsing text content line-by-line

When you have very large files, you probably do not want to bring the entire file content into memory and store it in a string

for loop line shredder

The entire file is fed into the *for* loop shredder...



Note: unlike a real shredder, the for loop shredder TM doesn't destroy your data—it just chops it into lines.

...which breaks it up into one-line-at-a-time chunks (which are themselves strings).

```
xfile = open('results.txt')
for line in xfile:
    print(line)
```

Reading file line-by-line

- A text file can be thought of as a *list of lines*
- A text file has newlines at the end of each line
- A file handle open for read can be treated as a **list of strings** where each line in the file is an element of this list
- We can use the **for** statement to iterate through a list of lines

```
xfile = open('results.txt')
for line in xfile:
    print(line)

line_list = [line for line in xfile]
    print(line_list)
```

One line at a time

All lines into a list of lines

Printing selected lines

- We can put an *if* statement in our *for* loop to only print lines that meet some criteria

```
fhand = open('emails.txt')
for line in fhand:
    if line.startswith('From:') :
        print(line)
```

OOPS!

- Why additional blank lines?

```
fhand = open('emails.txt')
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
```

Stripping newlines with *rstrip()*

- We can strip whitespaces from the right-hand side of the string using *rstrip()* method
- The newline is considered "white space" and is stripped

```
fhand = open('emails.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
```

Filtering lines using *continue*

```
fhand = open('emails.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:') :
        continue
    print(line)
```

```
fhand = open('emails.txt')
for line in fhand:
    line = line.rstrip()
    if not '@umich.edu' in line :
        continue
    print(line)
```

Find the highest score

```
f = open("results.txt")

max_score = 0
for line in f:
    score = float(line)
    if score > max_score:
        max_score = score

print(max_score)
```

Find top 3 highest scores

- Keeping track of 3 scores makes the logic more complex:

```
set the highest_score to 0
set the second_highest to 0
set the third_highest to 0
iterate through each of the scores:
if the score > highest_score:
    set the third_highest to second_highest
    set the second_highest to highest_score
    set the highest_score to score
otherwise if the score > second_highest:
    set the third_highest to second_highest
    set the second_highest to score
otherwise if the score > third_highest:
    set the third_highest to score
```

An ordered list would offer much simpler solution

- If you had some way of reading the data from the file and then producing **an ordered copy of the data**, the program would be a lot simpler to write
- Ordering data within a program is known as “sorting”
- We need to create a copy of disk data in memory **by adding every line to a list**

Useful list methods

Match each list method to the method description

1.	*count()*	A.	Sorts the list into a specified order (low to high)
2.	*extend()*	B.	Removes specified list item
3.	*index()*	C.	Adds an item at any index location
4.	*insert()*	D.	Looks for an item and returns its index value
5.	*pop()*	E.	Reverses the order of the list elements
6.	*remove()*	F.	Tells you how many times a value is in the list
7.	*reverse()*	G.	Adds a list of items to an original list
8.	*sort()*	H.	Removes and returns the first list item

Top 3 scores

```
f = open("results.txt")

sorted_scores = []
for line in f:
    score = float(line)
    sorted_scores.append(score)


sorted_scores.sort()
sorted_scores.reverse()
print(sorted_scores[0:3])
```

Who won the contest?

- Our top-3 program prints top 3 scores
- But who are the winners? Who should get a prize?
- How can we sort the scores and then find top-3 performers associated with high scores?

- Could we use parallel lists?

scores.txt

```
Johnny 8.65
Juan 9.12
Joseph 8.45
Stacey 7.81
Aideen 8.05
Zack 7.21
Aaron 8.31
```

Lists aren’t the only data structure to store data in memory

- We associate scores with names using a new data structure commonly called the hash table (better known as **dictionary** or associative array)

keys	values
8.65 $\longrightarrow$	Johnny
9.12 $\longrightarrow$	Juan
8.45	Joseph
7.81	Stacey
8.05	Aideen
7.21	Zack
8.31	Aaron

Summary

- Data in memory and in secondary storage
- Opening a file - file handle
- Manipulating file data through file handle: reading and writing
- File structure - newline character (and removing it with *rstrip*)
- Reading a file line-by-line with a for loop (or list comprehension)