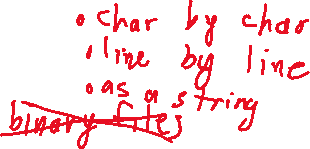
# Lecture 28 and 29

What are the two main types of files?



What’s the difference between a **folder** and a **directory**?



In Python, how can you find out the **current working directory** and **list the files** in a directory?

>>> import os

>>> os.getcwd()  
'C:\\Users\\james\\Documents\\courses\\cmpt120fall2022'

>>> os.listdir(os.getcwd())  
['a1.py', 'a2.py', 'lectures']

In Python, what’s a standard way to read a text file line-by-line?

textfile = open('joke.txt')  
for line in textfile:  
 print(line)

Who’s there?  
  
A broken pencil.  
  
A broken pencil who?  
  
Never mind. It’s pointless.

**What gets printed**

Who’s there?  
A broken pencil.  
A broken pencil who?  
Never mind. It’s pointless.

**Contents of joke.txt**

Why is the print-out **double-spaced**? How can you make it **single-spaced**?

textfile = open('joke.txt')  
for line in textfile:  
 print(line, end='') # don't put a \n after line

Or:

textfile = open('joke.txt')  
for line in textfile:  
 if line[-1] == '\n':  
 print(line[:-1]) # don't print the last character of line  
 else:  
 print(line)

Or:

def chop(s):  
 """If s ends with a \n, remove it. Otherwise returns s unchanged.  
 """  
 if s == '':   
 return ''  
 elif s[-1] == '\n':   
 return s[:-1] # returns all but last character of s  
 else:  
 return s

textfile = open('joke.txt')  
for line in textfile:  
 print(chop(line))

How can you **read lines of a text file into a list**?

Can also use the built-in method textfile.readlines()

textfile = open('joke.txt')  
all\_lines = []  
for line in textfile:  
 all\_lines.append(chop(line))   
  
print(all\_lines)  
print()  
print(f'number of lines: {len(all\_lines)}')

How could you put this into a function?

def get\_line\_list(fname):  
 textfile = open(fname)  
 all\_lines = []  
 for line in textfile:  
 all\_lines.append(chop(line))  
 return all\_lines

all\_lines = get\_line\_list('joke.txt')  
print(all\_lines)  
print()  
print(f'number of lines: {len(all\_lines)}')

**Practice** Using get\_line\_list, how could you:

* Print a file with line numbers?
* Print a file in reverse order?

**Challenge** Write a Python program that read the name of any Python source code file (which is a text file), and prints out just the names that are longer than 100 characters. Include the number of the line to help the user find the line in the file, and how long it is. Also, since the printed lines will be rather long, print only the first 20 or so characters followed by '…' to make sure they will be easily readable.

**Hints**

Use the file **test\_long\_lines.py** when we creating the program. It has lines of many different lengths, and will help us check that the program is working correctly, e.g.:

>>> print\_long\_lines('test\_long\_lines.py', 100)  
In "test\_long\_lines.py", these lines are over 100 characters:  
 line 5, 104 characters: " if n == 0 or ..."  
 line 11, 118 characters: "# This comment i ..."  
 line 17, 101 characters: "################ ..."

Every time you make a change, re-run the program on this test file to make sure it is correct.

Develop the program in “versions”. Start simple, and add one feature at a time:

* Version 1: read all the lines and print them.
* Version 2: add a line number to the start of each line.
* Version 3: only print lines greater than 100 characters.
* Version 4: print the long lines in a more compressed format.

What is a **data structure**? What are some of the data structures we’ve seen in Python?



What is the structure of Python’s **dictionary** data structure?



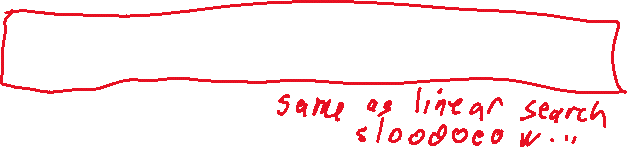
How do you read values by keys? How do you write values by keys?

What happens if you access a key that’s **not** in the dictionary?

Compared to **binary search on a sorted list**, is looking up a **key** in a dictionary usually faster or slower?



Compared to **binary search on a sorted list**, is looking up a **value** in a dictionary usually faster or slower?



Are dictionary **keys** unique? Are dictionary **values** unique?

**Challenge** Write a function called count\_words(fname) that returns a dictionary where the keys are words in the file, and the corresponding values are the count of the number of times the word occurs in the file.

Use the clean\_words function to remove unwanted characters from the lines:

def clean\_text(text):  
 """Remove all non-letter characters from text.

Keeps just alphabetic characters and spaces.

>>> clean\_text('Hello, world!')  
 'Hello world'  
 """  
 cleaned\_text = ''  
 for char in text:  
 if char.isalpha() or char == ' ':  
 cleaned\_text += char # keep spaces and letters  
 else:  
 cleaned\_text += ' ' # replace other characters with spaces  
 return cleaned\_text

If there’s time, write another function called print\_top10(fname) that uses the dictionary from count\_words(fname) to print the top 10 most frequently occurring words in fname.