Files

- ❖ Opening files
  afile = open(*filename*, *mode*)
  afile.*method*()
- ❖ Open mode
  - ➢ ‘r’ ➔ input
  - ➢ ‘w’ ➔ output
  - ➢ 'a‘ ➔ append
  - ➢ Adding a b to the mode string allows for *binary* data
  - ➢ Adding a + opens the file for *both* input and output
- ❖ Both of the first two arguments to open must be Python strings
- ❖ An optional third argument can be used to control output *buffering*—passing a zero means that output is unbuffered

1

Using Files

- ❖ *File iterators are best for reading lines*
- ❖ *Content is strings, not objects*
- ❖ *Files are buffered and seekable*
- ❖ close *is often optional: auto-close on collection*

```
>>> myfile = open('myfile.txt', 'w')       # Open for text output: create/empty
>>> myfile.write('hello text file\n')      # Write a line of text: string
16
>>> myfile.write('goodbye text file\n')
18
>>> myfile.close()                         # Flush output buffers to disk

>>> myfile = open('myfile.txt')            # Open for text input: 'r' is default
>>> myfile.readline()                      # Read the lines back
'hello text file\n'
>>> myfile.readline()
'goodbye text file\n'
>>> myfile.readline()                      # Empty string: end-of-file
''
```

- ❖ Notice that file write calls return the number of characters written

2

Using Files

- ❖ Write methods don’t add the end-of-line character for us
- ❖ Read the entire file into a string *all at once* with the file object’s read method

```
>>> open('myfile.txt').read()              # Read all at once into string
'hello text file\ngoodbye text file\n'

>>> print(open('myfile.txt').read())       # User-friendly display
hello text file
goodbye text file
```

- ❖ *File iterators* are often your best option

```
>>> for line in open('myfile.txt'):        # Use file iterators, not reads
...     print(line, end='')
...
hello text file
goodbye text file
```

3

Storing Python Objects in Files: Conversions

- ❖ Must convert objects to strings using conversion tools

```
>>> X, Y, Z = 43, 44, 45                   # Native Python objects
>>> S = 'Spam'                             # Must be strings to store in file
>>> D = {'a': 1, 'b': 2}
>>> L = [1, 2, 3]
>>>
>>> F = open('datafile.txt', 'w')          # Create output text file
>>> F.write(S + '\n')                      # Terminate lines with \n
>>> F.write('%s,%s,%s\n' % (X, Y, Z))      # Convert numbers to strings
>>> F.write(str(L) + '$' + str(D) + '\n')  # Convert and separate with $
>>> F.close()
```

- ❖ Notice that the interactive echo gives the exact byte contents, while the print operation interprets embedded endof- line characters to render a more user-friendly display:

```
>>> chars = open('datafile.txt').read()    # Raw string display
>>> chars
"Spam\n43,44,45\n[1, 2, 3]${'a': 1, 'b': 2}\n"
>>> print(chars)                           # User-friendly display
Spam
43,44,45
[1, 2, 3]${'a': 1, 'b': 2}
```

4

Storing Python Objects in Files: Conversions

- ❖ As Python never converts strings to numbers (or other types of objects) automatically, this is required if we need to gain access to normal object
- ❖ **rstrip** method to get rid of the trailing end-of-line character;

```
>>> F = open('datafile.txt')               # Open again
>>> line = F.readline()                    # Read one line
>>> line
'Spam\n'
>>> line.rstrip()                          # Remove end-of-line
'Spam'

>>> line = F.readline()                    # Next line from file
>>> line                                   # It's a string here
'43,44,45\n'
>>> parts = line.split(',')                # Split (parse) on commas
>>> parts
['43', '44', '45\n']

>>> int(parts[1])                          # Convert from string to int
44
>>> numbers = [int(P) for P in parts]      # Convert all in list at once
>>> numbers
[43, 44, 45]
```

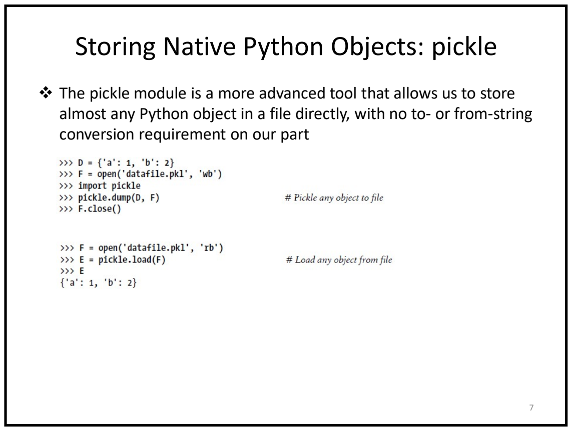
5

Storing Python Objects in Files: Conversions

- ❖ Using eval to convert from strings to objects

```
>>> line = F.readline()
>>> line
"[1, 2, 3]${'a': 1, 'b': 2}\n"
>>> parts = line.split('$')                # Split (parse) on $
>>> parts
['[1, 2, 3]', "{'a': 1, 'b': 2}\n"]
>>> eval(parts[0])                         # Convert to any object type
[1, 2, 3]
>>> objects = [eval(P) for P in parts]     # Do same for all in list
>>> objects
[[1, 2, 3], {'a': 1, 'b': 2}]
```

6



Storing Native Python Objects: pickle

- The pickle module is a more advanced tool that allows us to store almost any Python object in a file directly, with no to- or from-string conversion requirement on our part

```
>>> D = {'a': 1, 'b': 2}
>>> F = open('datafile.pkl', 'wb')
>>> import pickle
>>> pickle.dump(D, F)                          # Pickle any object to file
>>> F.close()


>>> F = open('datafile.pkl', 'rb')
>>> E = pickle.load(F)                         # Load any object from file
>>> E
{'a': 1, 'b': 2}
```

7