Text Data, File I/O, and Exceptions

Kenn H. Kim, Ph. D.

School of Business
Clemson University

Text Data, File I/O, and Exceptions

- Strings, Revisited
- Formatted Output
- File Input/Output

String Representations

A string value is represented as a sequence of characters delimited by quotes

Quotes can be single (') or double (")

What if the string includes both ' and "?

Escape sequence \' or \" is used to indicate that a quote is not the string delimiter but is part of the string value

Function `print()` interprets the escape sequence

Another example:

- `\n` is an escape sequence that represents a new line

```
>>> excuse = 'I am sick'
>>> excuse = "I am sick"
>>> excuse = 'I'm sick'
SyntaxError: invalid syntax
>>> excuse = "I'm sick"
>>> excuse = "I'm "sick""
SyntaxError: invalid syntax
>>> excuse = 'I'm "sick"'
SyntaxError: invalid syntax
>>> excuse = 'I\'m "sick"'
>>> excuse
'I\'m "sick"'
>>> print(excuse)
I'm "sick”
>>> excuse = 'I\'m ...\n... "sick"'
>>> excuse
'I\'m ...\n... "sick"'
>>> print(excuse)
I'm ...
... "sick"
```

Indexing Operator, Revisited

The indexing operator can also be used to obtain a slice of a string

`s[i:j]` : the slice of `s` starting at index `i` and ending before index `j`

`s[i:]` : the slice of `s` starting at index `i`

`s[:j]` : the slice of `s` ending before index `j`

```
              -5   -4   -3   -2   -1
s         =  ' A    p    p    l    e '
              0    1    2    3    4
s[0:2]    =  'A    p'
s[1:4]    =       'p    p    l'
s[2:5]    =            'p    l    e'
s[2:]     =            'p    l    e'
s[:2]     =  'A    p'
s[-3:-1]  =            'p    l'
```

```
>>> s = 'Apple'
>>> s[0:2]
'Ap'
>>> s[1:4]
'ppl'
>>> s[2:5]
'ple'
>>> s[2:]
'ple'
>>> s[:2]
'Ap'
>>> s[-3:-1]
'pl'
```

Exercise (Notebook)

The indexing operator can also be used to obtain slices of a list as well. Let list `lst` refer to list

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h’]
```

Write Python expressions using list `lst` and the indexing operator that evaluate to:

```
a) ['a', 'b', 'c', 'd']
b) ['d', 'e', 'f']
c) ['d']
d) ['f', 'g']
e) ['d', 'e', 'f', 'g’, 'h']
f) ['f', 'g', 'h']
```

String Methods

Strings are immutable; none of the string methods modify string `link`

Usage	Explanation
`s.capitalize()`	returns a copy of `s` with first character capitalized
`s.count(target)`	returns the number of occurrences of `target` in `s`
`s.find(target)`	returns the index of the first occurrence of `target` in `s`
`s.lower()`	returns lowercase copy of `s`
`s.upper()`	returns uppercase copy of `s`
`s.split(sep)`	returns list of substrings of `s`, delimited by `sep`

String Methods

Strings are immutable; none of the string methods modify string `link`

```
>>> link = 'http://www.main.com/smith/index.html'
>>> link[:4]
'http'
>>> link[:4].upper()
'HTTP'
>>> link.find('smith')
20
>>> link[20:25]
'smith'
>>> link[20:25].capitalize()
'Smith'
>>> link
'http://www.main.com/smith/index.html'
>>> link.count('/')
4
>>> link.split('/')
['http:', '', 'www.main.com', 'smith', 'index.html']
```

Exercise (Notebook)

```
>>> events = '9/13 2:30 PM\n9/14 11:15 AM\n9/14 1:00 PM\n9/15 9:00 AM'
>>> print(events)
9/13 2:30 PM
9/14 11:15 AM
9/14 1:00 PM
9/15 9:00 AM
```

String `events` describes the schedule of 4 events spread across 3 days

Write expressions that compute:

a) the number of events on 9/14

b) the index of the substring describing the 1st event on 9/14

c) the index just past the substring describing the last event on 9/14

d) the list of substrings describing the events on 9/14

Built-in Function `print()`, Revisited

CLEMSON College of BUSINESS

Function `print()` prints, by default, a newline character after printing its arguments

```
>>> pets = ['boa', 'cat', 'dog']
>>> for pet in pets:
        print(pet)


boa
cat
dog
>>> for pet in pets:
        print(pet, end=', ')


boa, cat, dog,
>>> for pet in pets:
        print(pet, end='!!! ')


boa!!! cat!!! dog!!!
>>>
```

The `end` argument allows for customized end characters

General Output Formatting

CLEMSON College of BUSINESS

Suppose we have

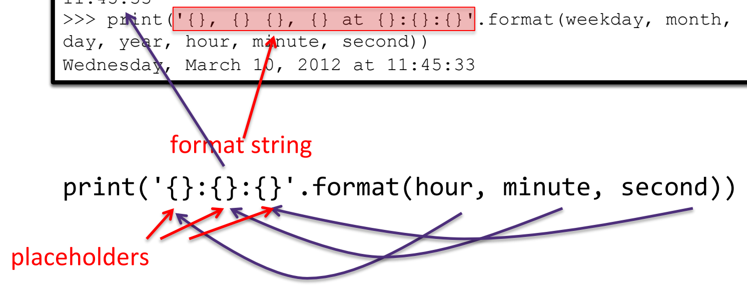
```
>>> weekday = 'Wednesday'
>>> month = 'March'
>>> day = 10
>>> year = 2010
>>> hour = 11
>>> minute = 45
>>> second = 33
>>> print(hour+':'+minute+':'+second)
Traceback (most recent call last):
  File "<pyshell#113>", line 1, in <module>
    print(hour+':'+minute+':'+second)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> print(str(hour)+':'+str(minute)+':'+str(second))
11:45:33
>>> print('{}:{}:{}'.format(hour, minute, second))
11:45:33
```

and we want to print `Wednesday, March 10, 2010 at 11:45:33`

Method `format()` of Class `str`

CLEMSON College of BUSINESS

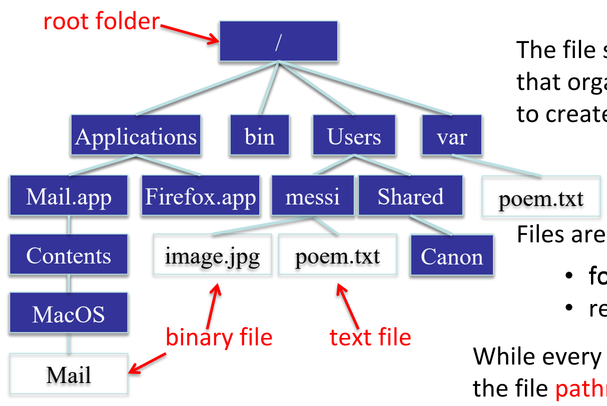
```
>>> day = 'Wednesday'
>>> month = 'March'
>>> weekday = 'Wednesday'
>>> month = 'March'
>>> day = 10
>>> year = 2010
>>> year = 2012
>>> hour = 11
>>> minute = 45
>>> second = 33
>>> print('{}:{}:{}'.format(hour, minute, second))
11:45:33
>>> print('{}, {} {}, {} at {}:{}:{}'.format(weekday, month,
day, year, hour, minute, second))
Wednesday, March 10, 2012 at 11:45:33
```



```
print('{}:{}:{}'.format(hour, minute, second))
```

Files and the File System

CLEMSON College of BUSINESS



The file system is the OS component that organizes files and provides a way to create, access, and modify files

Files are organized into a tree structure

- folders (or directories)
- regular files

While every file and folder has a name, it is the file pathname that identifies the file

Absolute pathnames

- `/var/poem.txt`
- `/Users/messi/poem.txt`
- `/Applications/Mail.app/`

Relative pathnames
(relative to current working directory `Users`)

- `messi/poem.txt`
- `messi/image.jpg`
- `Shared`

Opening and Closing a File

Processing a file consists of:

1. Opening the file
2. Reading from and/or writing to the file
3. Closing the file

File mode `'r'` is used to open a file for reading (rather than, say, writing)

Built-in function `open()` is used to open a file

- The first input argument is the file pathname, whether absolute or relative with respect to the current working directory
- The second (optional) argument is the file mode
- Returns a "file" object

A "file" object is of a type that supports several "file" methods, including method `close()` that closes the file

```
>>> infile = open('sample.txt')
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    infile = open('sample.txt')
IOError: [Errno 2] No such file or directory:
'sample.txt'
>>> infile = open('example.txt', 'r')
>>> infile.close()
>>>
```

Open File Mode

The file mode defines how the file will be accessed

Mode	Description
`r`	Reading (default)
`w`	Writing (if file exists, content is wiped)
`a`	Append (if file exists, writes are appended)
`r+`	Reading and Writing
`t`	Text (default)
`b`	Binary

These are all equivalent ⟶

```
>>> infile = open('example.txt', 'rt')
>>> infile = open('example.txt', 'r')
>>> infile = open('example.txt', 't')
>>> infile = open('example.txt')
```

File Methods

There are several “file” types; they all support similar “file” methods

- Methods `read()` and `readline()` return the characters read as a string
- Methods `readlines()` returns the characters read as a list of lines
- Method `write()` returns the number of characters written

Usage	Description
`infile.read(n)`	Read `n` characters starting from cursor; if fewer than `n` characters remain, read until the end of file
`infile.read()`	Read starting from cursor up to the end of the file
`infile.readline()`	Read starting from cursor up to, and including, the end of line character
`infile.readlines()`	Read starting from cursor up to the end of the file and return list of lines
`outfile.write(s)`	Write string `s` to file `outfile` starting from cursor
`infile.close(n)`	Close file `infile`

Reading a File

```
The 3 lines in this file end with the new line character.\n
\n
There is a blank line above this line.\n
```

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is `r`
- at the end of the file, if file mode is `a` or `w`

```
>>> infile = open('example.txt')
>>> infile.read(1)
'T'
>>> infile.read(5)
'he 3 '
>>> infile.readline()
'lines in this file end with the new line
character.\n'
>>> infile.read()
'\nThere is a blank line above this line.\n'
>>> infile.close()
>>>
```

Exercise (Notebook)

Patterns for Reading a Text File

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numChars(filename):
    'returns the number of characters in file filename'

    infile = open(filename, 'r')
    content = infile.read()
    infile.close()

    return len(content)
```

Exercise (Notebook)

Patterns for Reading a Text File

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numWords(filename):
    'returns the number of words in file filename'

    infile = open(filename)
    content = infile.read()
    infile.close()
    wordList = content.split()

    return len(wordList)
```

Exercise (Notebook)

Patterns for Reading a Text File

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numLines(filename):
    'returns the number of lines in file filename'

    infile = open(filename, 'r’)
    lineList = infile.readlines()
    infile.close()

    return len(lineList)
```

Writing to a Text File

```
This is the first line. Still the first line…\n
Now we are in the second line.\n
Non string value like 5 must be converted first.\n
Non string value like 5 must be converted first.\n
```

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>> outfile.write(' Still the first line...\n')
25
>>> outfile.write('Now we are in the second line.\n')
31
>>> outfile.write('Non string value like '+str(5)+' must be converted first.\n')
49
>>> outfile.write('Non string value like {} must be converted first.\n'.format(5))
49
>>> outfile.close()
```

Exercise (Notebook)

Writing to a Text File “new_file.txt”

Hello, World!

I love python programming.

Really?

We covered
Text Data, File I/O, and Exceptions

- Strings, Revisited
- Formatted Output
- File Input/Output