

Files

Topic 9

Dr. Douglas Stebila

Department of Computing and Software
McMaster University

Spring/Summer 2020



Table of Contents

Intro

File formats

Intro

File formats

input and print

[prompt] means prompt is optional

- ▶ `input([prompt])`
 - If prompt present, it is printed to the screen. A line is read from the user and returned as a string.
- ▶ `print(object, ..., sep=' ', end='\n')`
 - Converts all objects with str to strings and prints them to the screen, separated by sep (' ' by default) and terminated by end ('\n' by default)
- ▶ `>>> print(1, 2, 3)`
- ▶ `>>> print(1, 2, 3, sep = ' ')`
- ▶ `>>> print(1, 2, 3, end = '\n\n')`

Standard input / standard output

- ▶ Printing to the screen and reading from the user are modelled by special files called standard output (`sys.stdout`) and standard input (`sys.stdin`)
- ▶ In command-line scripts, we can redirect standard input and standard output to files
 - Outside the scope of this course

input and print

- ▶ `input([prompt])`
 - If prompt present, it is written to **standard output**. A line is read from **standard input** and returned as a string.

- ▶ `print(object, ..., sep=' ', end='\n')`
 - Converts all objects with `str` to strings and writes them to **standard output**, separated by `sep` (' ' by default) and terminated by `end` ('\n' by default)

- ▶ `>>> print(1, 2, 3)`
- ▶ `>>> print(1, 2, 3, sep = ' ')`
- ▶ `>>> print(1, 2, 3, end = '\n\n')`

Basic procedure for working with files

1. Open the file for reading or writing to obtain a “file handle”.
 - Can only open file exclusively for reading or exclusively for writing at one time – can’t read and write to the same file at the same time.
2. Read or write one or more lines to the file by referring to its file handle.
3. Close the file.

Reading an entire file into a string

```
fh = open('myfile.txt', 'r')
s = fh.read()
fh.close()

print("The file contained:")
print(s)
```


Functions for reading text files

```
fh = open(filename, mode='r')
```

Opens `filename` and returns a file object `fh`, `mode` specifies the mode: `r` for reading, `w` for writing, `a` for appending

```
fh.read()
```

Reads the entire file to memory and returns it as a string

```
fh.readline()
```

Reads the next line to memory and returns it as a string.
(The string includes the newline character.) If the end of the file is reached, the empty string is returned

```
fh.close()
```

Closes the file and frees the resources associated with the file object.

Reading an entire file into a string using “with”

```
with open('myfile.txt', 'r') as fh:
    s = fh.read()

# "with" automatically closes fh
# so no need for fh.close()

print("The file contained:")
print(s)
```

Reading a file line by line

```
with open('myfile.txt', 'r') as fh:
    line = fh.readline()
    while line != "":
        print("Read the line:")
        print(line)
        line = fh.readline()
```

Reading a file line by line

```
with open('myfile.txt', 'r') as fh:
    for line in fh:
        print("Read the line:")
        print(line)
```

- ▶ line will include the end-of-line character '\n'
- ▶ It can be removed using the `rstrip()` method:
 - `line = line.rstrip()`

Handling exceptions

```
try:
    with open('missing.txt', 'r') as fh:
        for line in fh:
            print(line)
except FileNotFoundError:
    print("File not found")
```

Writing to a file

```
with open('myfile.txt', 'w') as fh:  
    fh.write("This is a test")
```

Functions for writing text files

```
fh = open(filename, mode='r')
```

Opens `filename` and returns a file object `fh`, `mode` specifies the mode: `r` for reading, `w` for writing, `a` for appending

```
fh.read()
```

Writes the string `s` to the file

```
fh.readline()
```

Converts all objects with `str` to strings and writes them to `fh`, separated by `sep` (' ' by default) and terminated by `end` ('\n' by default)

```
fh.close()
```

Closes the file and frees the resources associated with the file object.

Writing to a file

```
with open('myout.txt', 'w') as fh:  
    print("A string", file=fh)  
    print("Another string", file=fh)  
    print(17, file=fh)
```


Writing to standard out using file handles

```
import sys
sys.stdout.write("A string")
```

Reading data from the Internet

- ▶ Same idea as reading data from a file, except our file handle is the file handle of a remote resource rather than a local file
- ▶ We'll use the urllib module to open remote resources

Reading data from the Internet

```
import urllib.request
url = 'https://www.douglas.stebila.ca/cs1md3.txt'
response = urllib.request.urlopen(url)
s = response.read()
```

```
print(s)
```

```
b'This is a test file for CS 1MD3.\nIt contains 2 lines.\n'
```

```
print(type(s))
```

```
<class 'bytes'>
```

- Data read from a URL is a sequence of bytes.
- It could be binary data or it could represent a string.
- If it represents a string, special characters (accents, emoji, ...) could be encoded in one of many different character sets.
- So we need to decode it.

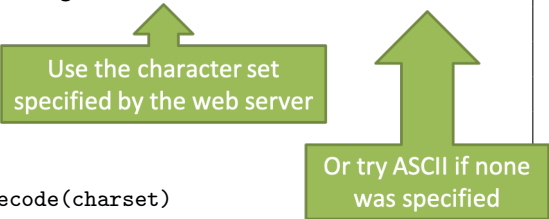
Reading data from the Internet

```
import urllib.request
url = 'https://www.douglas.stebila.ca/cs1md3.txt'
response = urllib.request.urlopen(url)

charset = response.info().get_content_charset() or 'ascii'

s = response.read().decode(charset)

print(s)
print(type(s))
```



Use the character set specified by the web server

Or try ASCII if none was specified

Reading data from the Internet line by line

```
import urllib.request
url = 'https://www.douglas.stebila.ca/cs1md3.txt'
response = urllib.request.urlopen(url)
charset = response.info().get_content_charset() or 'ascii'
for line in response:
    s = line.decode(charset)
    print(s)
```

Handling errors

```
import urllib.request
from urllib.error import HTTPError
url = 'https://www.douglas.stebila.ca/missing.txt'

try:
    response = urllib.request.urlopen(url)
    # do whatever
except HTTPError:
    print("Could not load the resource")
```

Table of Contents

Intro

File formats

CSV – Comma-separated values

- ▶ A plain-text format for storing tabular data
- ▶ Commonly used for importing/exporting spreadsheets between applications
 - E.g. we export CSV grades from JupyterHub, import them into Avenue, then at the end of semester export CSV from Avenue, import into Excel to do the final calculations, then export to CSV and import into Mosaic

CSV – Comma-separated values

- ▶ Each line of the file is a data record
- ▶ Each record of the file contains multiple fields separated by commas
- ▶ Can optionally have a header row containing field names
- ▶ No single standard definition of the CSV format, so sometimes have compatibility problems

CSV example file

Last Name	First Name	MacID	A1	A2
Chen	Alice	chena	17	15
Jones	Bob	jonesb	16	16
Smith	Carol	smithc	15	17

```
Last Name,First Name,MacID,A1,A2
Chen,Alice,chena,17,15
Jones,Bob,jonesb,16,16
Smith,Carol,smithc,15,17
```



Optional
header
row

Reading CSV in Python

The csv module contains functions for working with CSV files

1. The csv module contains functions for working with CSV files
2. Create a CSV reader object based on the file handle
3. Use a for loop to iterate through the CSV reader object; each line is given as a list
 - If the first line is a header row, we have to put in extra logic to skip it

```
import csv
with open('grades.csv', 'r') as fh:
    grades = csv.reader(fh)
    for row in grades:
        print(row)
```

Reading CSV in Python

If the first line of the CSV file is a header row, we can use `csv.DictReader` instead of `csv.reader` to get each row as a dictionary

```
import csv
with open('grades.csv', 'r') as fh:
    grades = csv.DictReader(fh)
    for row in grades:
        print(row["First Name"])
```

Writing CSV in Python

1. Open the file (as in previous lectures)
2. Create a CSV write object based on the file handle
3. Use the writerow method one at a time to write a list out as a row

```
import csv
with open('grades2.csv', 'w') as fh:
    grades = csv.writer(fh)
    grades.writerow(['chena', 100])
    grades.writerow(['jonesb', 100])
```

Writing CSV in Python

If your data is already represented as a list of lists, then you can write it out all at once.

```
import csv
L = [ ['chena', 100], ['jonesb', 100] ]
with open('grades3.csv', 'w') as fh:
    grades = csv.writer(fh)
    grades.writerows(L)
```

Complications

- ▶ What if one of your fields contains a comma?
 - It should be wrapped in quotes
 - Python's CSV reader/writer will automatically handle this

```
Last Name,First Name,MacID,A1,A2
Chen,"Alice, the Princess of Wales",chena,17,15
Jones,Bob,jonesb,16,16
Smith,Carol,smithc,15,17
```

Complications

- ▶ Accented characters / other alphabets / emoji can be tricky
 - Especially when trying to exchange between Excel and other programs
- ▶ Sometimes people use other “delimiters” instead of commas
 - Tabs (“tab-separated value”)
 - | (vertical bar)
 - spaces

CSV data sources

- ▶ Can be exported from any Excel/OpenOffice/LibreOffice spreadsheet
 - But no macros/formulas are exported – only the resulting values
- ▶ Many financial websites
 - E.g., Yahoo Finance
<https://finance.yahoo.com/quote/AAPL/history?p=AAPL>
→ Download data

JSON – JavaScript Object Notation

- ▶ Originally a subset of the JavaScript programming language for representing objects
- ▶ Now a language-independent plain-text format for representing dictionaries and lists
- ▶ Commonly used for transmitting objects between applications

JSON – JavaScript Object Notation

- ▶ Basic format: the overall file is either a list or a dictionary
- ▶ Every entry can be one of the following:
 - List (enclosed in [...])
 - Dictionary
 - Enclosed in { ... }
 - Every key is a quoted string
 - Colon between key and value
 - Value can be any kind of entry
 - Number (int or float)
 - String (enclosed in " ... ")
 - Boolean (true or false)
 - null

JSON example file

```
[  
  { "name": "Harry Potter", "year": 1980,  
    "spouse": "Ginny Weasley",  
    "children": ["James", "Albus", "Lily"] },  
  { "name": "Hermione Granger", "year": 1979,  
    "spouse": "Ron Weasley" },  
  { "name": "Tom Riddle", "year": 1926,  
    "spouse": null }  
]
```

Whitespace (spaces, tabs, line breaks) are irrelevant in JSON files

Reading JSON in Python

The json module contains functions for working with JSON data

1. Get the JSON data as a string
 - Possible by opening a file and then reading from the file handle
2. Use the json.loads to parse the JSON data into a list or dictionary
3. Access the list or dictionary normally in Python

```
import json
with open('potter.json', 'r') as fh:
    s = fh.read()
    x = json.loads(s)
    print("Got a", type(x), "of length", len(x))
```

Writing JSON in Python

- ▶ You can convert any list or dictionary into a JSON string using the `json.dumps` function
- ▶ If you want you can then write this to a file in the normal way

```
import json
d = {}
d["studio"] = "Disney"
d["movies"] = ["Beauty and the Beast", "Lion King"]
with open('animated.json', 'w') as fh:
    s = json.dumps(d)
    fh.write(s)
```

JSON data sources

Many websites will provide some access to their data in an automated way (via a “web application programming interface (API)”)

- ▶ Data returned from web APIs often in JSON format
- ▶ Examples:
 - Facebook
 - Instagram
 - Twitter

JSON example — Instagram

```
{
  "data": [
    {
      "caption": {
        "created_time": "1296710352",
        "text": "Inside le truc #foodtruck",
        "from": { "username": "kevin", "full_name": "Kevin Systrom" },
      },
      "likes": { "count": 15 },
      "link": "http://instagr.am/p/B6rVZ/",
      "created_time": "1296710327",
      "images": {
        "low_resolution": {
          "url": "http://distillery.s3.amazonaws.com/media/2011/02/02/6ea7baea55774c5e81e7e3e1f6e791a7_6.jpg",
          "width": 306, "height": 306
        },
        "thumbnail": {
          "url": "http://distillery.s3.amazonaws.com/media/2011/02/02/6ea7baea55774c5e81e7e3e1f6e791a7_5.jpg",
          "width": 150, "height": 150
        },
        "standard_resolution": {
          "url": "http://distillery.s3.amazonaws.com/media/2011/02/02/6ea7baea55774c5e81e7e3e1f6e791a7_7.jpg",
          "width": 612, "height": 612
        }
      },
      "type": "image",
      "filter": "Earlybird",
      "tags": ["foodtruck"],
    },
    {
      "videos": {
        "low_resolution": {
          "url": "http://distilleryvesper9-13.ak.instagram.com/090d86dad9cd11e2aa0912313817975d_102.mp4",
          "width": 480, "height": 480
        },
        "standard_resolution": {
          "url": "http://distilleryvesper9-13.ak.instagram.com/090d86dad9cd11e2aa0912313817975d_101.mp4",
          "width": 640, "height": 640
        },
        "caption": null,
        "likes": { "count": 1 },
        "link": "http://instagr.am/p/D/",
        "created_time": "1279340983",
        "type": "video",
        "filter": "Vesper",
        "tags": [],
      },
    },
  ],
}
```

<https://www.instagram.com/developer/endpoints/users/>

XML – Extensible Markup Language

- ▶ General-purpose plain text format for formatting data that is meant to be both machine-readable and human-readable
- ▶ Hierarchical structure
- ▶ Commonly used for transmitting objects between applications
- ▶ Somewhat difficult to program with

I will never make you
work with XML files. This
is just for your interest.

XML – Extensible Markup Language

- ▶ Basic format:
 - ▶ Every element is wrapped inside a “tag” denoting the type of element it is
 - ▶ `<tagname>` to start
 - ▶ `</tagname>` to end
 - ▶ Elements can be nested inside each other
 - ▶ Elements can have “attributes” added onto their “tag” definition
 - ▶ Whitespace doesn't matter

XML example file

```
<familytree>
  <person born="1980">
    <name>Harry Potter</name>
    <children>
      <person born="2004">
        <name>James Potter</name>
      </person>
      <person born="2006">
        <name>Albus Potter</name>
      </person>
      <person born="2008">
        <name>Lily Potter</name>
      </person>
    </children>
  </person>
</familytree>
```

Reading XML in Python

- ▶ Reading and working with XML is actually quite tricky
 - Lots of "extra stuff" comes out when parsing an XML file
- ▶ Can be hard to pull out exactly what you're looking for because of the extra stuff and the difficulty of precisely indicating a particular point in the XML file

Reading XML in python

```
import xml.dom.minidom
with open('potter.xml') as fh:
    s = fh.read()
    x = xml.dom.minidom.parseString(s)
    people = x.getElementsByTagName("person")
    for person in people:
        name = person.getElementsByTagName("name")[0]
        print(name.childNodes[0].data)
```

Harry Potter
James Potter
Albus Potter
Lily Potter

Reading XML in Python

```
import xml.dom.minidom
with open('potter.xml') as fh:
    s = fh.read()
    x = xml.dom.minidom.parseString(s)
    people = x.getElementsByTagName("person")
    for person in people:
        name = person.getElementsByTagName("name")[0]
        print(name.childNodes[0].data)
```

- ▶ `xml.dom.minidom.parseString` parses the XML string into an object
- ▶ `getElementsByTagName` returns an array containing all of the children

XML data sources

- ▶ Lots of websites make a "news feed" of their recent articles available in an XML-based format called RSS or ATOM E.g.
<https://dailynews.mcmaster.ca/feed/>
- ▶ Many business and government oriented sites make data available in XML format
 - E.g. Government of Canada
https://weather.gc.ca/rss/city/on-77_e.xml
https://www.bankofcanada.ca/valet/fx_rss/

XML example — Current Bank of Canada USD-CAD exchange rates

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
  xmlns:cb="http://www.cbwiki.net/wiki/index.php/Specification_1.1"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  <channel rdf:about="https://www.bankofcanada.ca/valet/fx_rss/FXUSDCAD">
    <title>Daily exchange rates</title>
    <link>https://www.bankofcanada.ca/?p=39898</link>
    <description>Daily average exchange rates - published once each business day by 16:30 ET. All Bank of Canada exchange rates are indicative rates only.</description>
  </channel>
  <item rdf:about="https://www.bankofcanada.ca/valet/fx_rss/FXUSDCAD">
    <title>CA: 1.2927 CAD = 1 USD 2018-03-08</title>
    <link>https://www.bankofcanada.ca/?p=39898</link>
    <description>1 USD = 1.2927 CAD (US dollar to Canadian dollar daily exchange rate)</description>
    <dc:date>2018-03-08T21:30:00Z</dc:date>
    <dc:language>en</dc:language>
    <cb:statistics>
      <cb:country>CA</cb:country>
      <cb:exchangeRate>
        <cb:value decimals="4">1.2927</cb:value>
        <cb:baseCurrency>CAD</cb:baseCurrency>
        <cb:targetCurrency>USD</cb:targetCurrency>
        <cb:rateType>Bank of Canada exchange rate</cb:rateType>
        <cb:observationPeriod frequency="daily">2018-03-08T21:30:00Z</cb:observationPeriod>
      </cb:exchangeRate>
    </cb:statistics>
  </item>
</rdf:RDF>
```

https://www.bankofcanada.ca/valet/fx_rss/FXUSDCAD

XML example – Current Bank of Canada USD-CAD exchange rates

```
import xml.dom.minidom
import urllib.request

# download the XML data from the web site
url = 'https://www.bankofcanada.ca/valet/fx_rss/FXUSDCAD'
response = urllib.request.urlopen(url)
charset = response.info().get_content_charset() or 'ascii'
s = response.read().decode(charset)

# parse the XML
x = xml.dom.minidom.parseString(s)
# pull out the exact element we want
rate = x.getElementsByTagName("cb:value")[0].childNodes[0].data

print(rate)
```

HTML – Hypertext Markup Language

- ▶ Plaintext language used for describing the content of web pages
- ▶ Every page you browse on the web is an HTML file
- ▶ Hierarchical structure
- ▶ Similar to XML
- ▶ Possible to extract data from HTML files ("scraping"), but this can be tricky since websites change their HTML source code frequently
 - Preferable to use CSV/JSON/XML if available