Getting data

Khue Do

IM, VAST

Table of contents

Motivation

- In order to be a **data** scientist you need data.
- **Where** to acquire data?
- How to get data **efficiently**?
- How to extract **good information** from the acquired data?

I/O Stream

I/O Stream

Stream I/O refers to the transfer of data either to or from a storage medium. Streams are a mean to manipulate the data that is read or written from and to a file.

- **Input Stream:** If the direction of flow is from device to the memory.
- **Output Stream:** If the direction of flow is opposite

Standard I/O

Standard Streams

Standard Streams are a feature of many operating systems:

- Read input from the keyboard.
- Write output to the display.

Standard I/O

Standard I/O

- standard input **(stdin)** - Read from the keyboard or from a **redirection** such as **the pipe**.
- standard output **(stdout)** - Print to the display and can be **redirected** as standard input.

Pipeline

Pipes

Pipes connect the standard output of one command to the standard input of another.

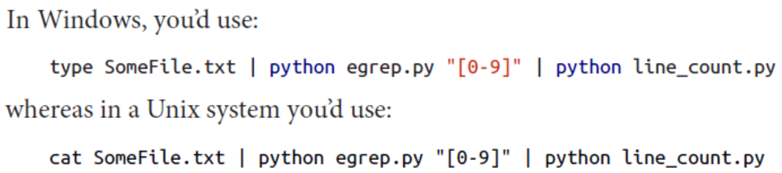
Standard I/O in Python

Several ways to do it:

- **sys.stdin** and **sys.stdout** are file-like objects on which you can call functions **read()** or **write()**.
- If you want to prompt the user for input, you can use **input()** in Python 3 (**raw_input()** for Python 2).
- If you actually just want to read command-line options, you can access them via the **sys.argv** list.

Pipe character in Python

How to use pipes:



In Windows, you'd use:

```
type SomeFile.txt | python egrep.py "[0-9]" | python line_count.py
```

whereas in a Unix system you'd use:

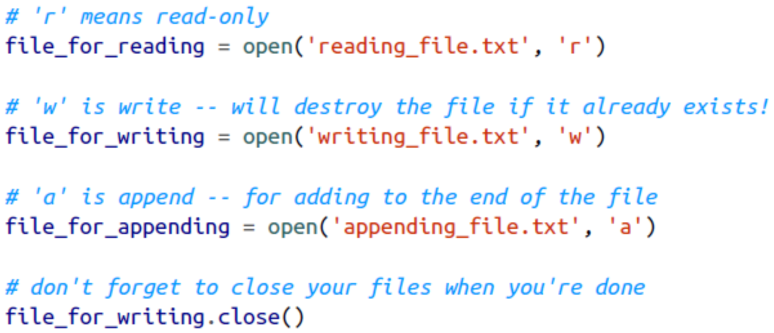
```
cat SomeFile.txt | python egrep.py "[0-9]" | python line_count.py
```

Remark: Pipe need not to be misused of the **OR operator**.

Handling pipeline with FP

- Are chains of I/O.
- **Edge case** for each Input must be handle (permission to read, file not exist, . . .).
- Pipe break can be handled without catching exception by using functional programming.

I/O Stream via files in Python



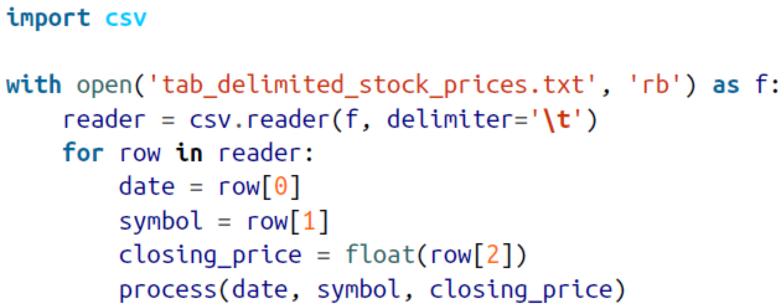
```
# 'r' means read-only
file_for_reading = open('reading_file.txt', 'r')

# 'w' is write -- will destroy the file if it already exists!
file_for_writing = open('writing_file.txt', 'w')

# 'a' is append -- for adding to the end of the file
file_for_appending = open('appending_file.txt', 'a')

# don't forget to close your files when you're done
file_for_writing.close()
```

Delimited Files

- Data file should be represented in a table.
- Format of the files should be considered.
- Suggested to work with **CSV** file.

Example of delimiting files

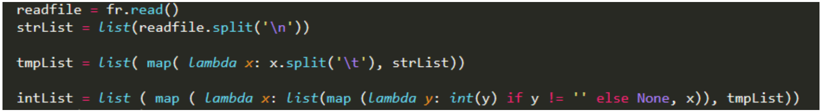
Example of delimiting data:



```
import csv

with open('tab_delimited_stock_prices.txt', 'rb') as f:
    reader = csv.reader(f, delimiter='\t')
    for row in reader:
        date = row[0]
        symbol = row[1]
        closing_price = float(row[2])
        process(date, symbol, closing_price)
```

Example of delimiting files using FP

Example of using functional programming to delimit data:



```
readfile = fr.read()
strList = list(readfile.split('\n'))

tmpList = list( map( lambda x: x.split('\t'), strList))

intList = list ( map ( lambda x: list(map (lambda y: int(y) if y != '' else None, x)), tmpList))
```

JSON

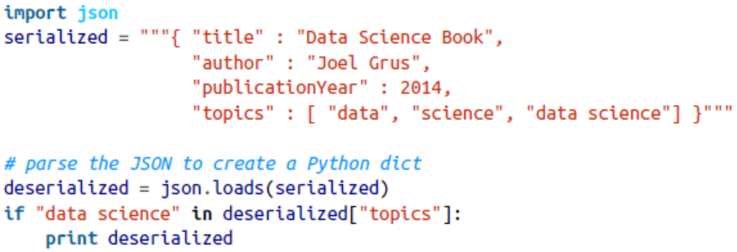
- Data requested through a web API needs to be serialized into a string format and often is **JSON**.
- **JSON** stand for JavaScript Object Notation.
- Quite similar to Python dicts.

An example of JSON using Python’s json module

An example of JSON:

```
{ "title" : "Data Science Book",
  "author" : "Joel Grus",
  "publicationYear" : 2014,
  "topics" : [ "data", "science", "data science"] }
```

An example of parsing JSON



```
import json
serialized = """{ "title" : "Data Science Book",
                  "author" : "Joel Grus",
                  "publicationYear" : 2014,
                  "topics" : [ "data", "science", "data science"] }"""

# parse the JSON to create a Python dict
deserialized = json.loads(serialized)
if "data science" in deserialized["topics"]:
    print deserialized
```

Finding APIs

- Look for a developers or API section of the site for details, and try searching the Web for "python␣api" to find a library.
- If you're looking for lists of APIs that have Python wrappers, two directories are at **Python API** and **Python for Beginners**.
- If you want a directory of web APIs more broadly, a good resource is **Programmable Web**, which has a huge directory of categorized APIs.