Ulster University

COM738 - Data Science Foundations

Week 3_Part 2

Handling Data

Module Co-Ordinator: Dr Priyanka Chaurasia

Week 3 - Contents

- Data Wrangling
- Data Formats
  - CSV, XML, JSON
- Python Modules
- Reading Data from Files
- Writing Data to Files
- Analyse the contents of a file, and create a new file with additional metrics

Ulster University

Data Wrangling

- Aka. "Data Munging"
- The process of **cleaning and formatting data**, transforming it from its raw form into a format that is more usable for further analysis

"*Most data scientists spend much of their time cleaning and formatting data. The rest spend most of their time complaining that there is no available data to do what they want to do!*"

SSSkiena, The Data Science Design Manual (2017)

- Garbage In = Garbage Out

Ulster University

Dirty Data Sample

	DBAName	AKAName	Address	City	State	Zip
t1	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	***Chicago***	IL	***60608***
t2	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	***60609***
t3	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	***60609***
t4	***Johnnyo's***	Johnnyo's	3465 S Morgan ST	***Cicago***	IL	60608

Conflicts

Does not obey data distribution

Conflict

Ulster University

What is a “good” data format?

Ulster University

Data Formats

Good data formats could be:

- **Easy for computers to parse** – A logical and consistent format
  - **Easy for people to read** – "Eyeballing" data can be an essential operation in many scenarios. E.g.
    - Which of the files in this directory is the right one?
    - What information do these data fields really contain?
    - What range of values can be expected from a particular field?
    - This may be through use of clear delimiting symbols, human-readable text-encoded formats, etc.
  - **Widely used by other tools and systems** -
    - This allows mixing and matching data from various resources, best facilitated through using popular standard formats

Ulster University

Data Formats

Most common data formats:

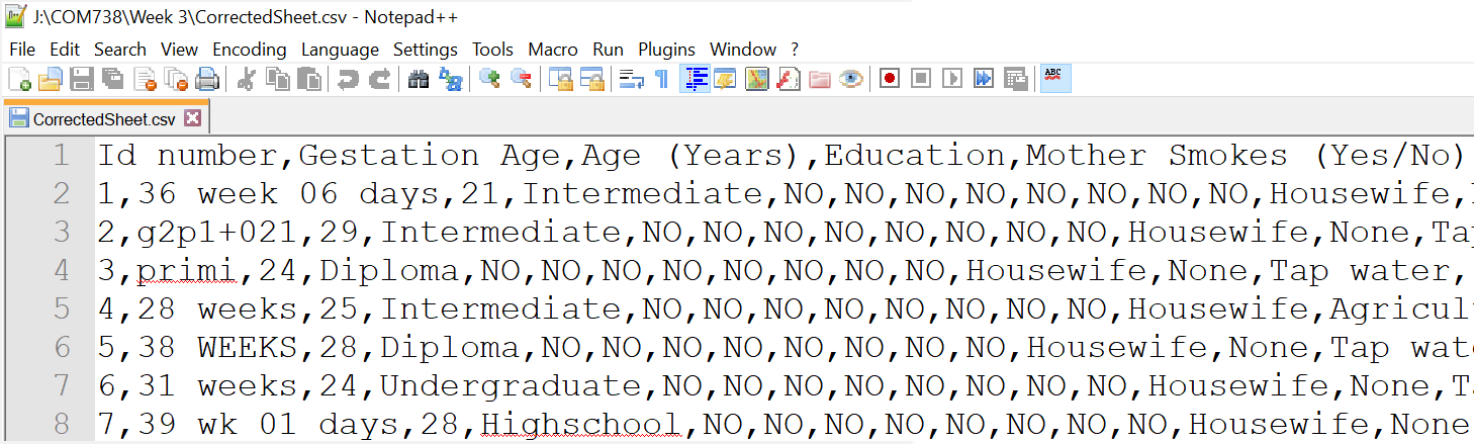
- CSV– Comma Separated Value
- XML – eXtensible Markup Language
- JSON- JavaScript Object Notation
- Database

Ulster University

Data Formats

CSV– Comma Separated Value

- Field names in the first row
- Each value separated by a comma (or other delimiter)
  - **Delimiter** – one or more characters used to specify the boundary between separate regions in data
- Each row separated by a carriage return
- Notepad++ for eyeballing (or a spreadsheet application)



Ulster University

Data Formats

CSV– Comma Separated Value

- Notepad++ for eyeballing (or a spreadsheet application)
  - Notepad++ can show all characters

```
policyID,statecode,county,eq_site_limit,hu_site_limit,fl_site_limit,fr_site_limit,tiv_2011,tiv_2012,eq_site_deduct
119736,FL,CLAY COUNTY,498960,498960,498960,498960,498960,792148.9,0,9979.2,0,0,30.102261,-81.711777,Residential,Ma
448094,FL,CLAY COUNTY,1322376.3,1322376.3,1322376.3,1322376.3,1322376.3,1438163.57,0,0,0,0,30.063936,-81.707664,Re
206893,FL,CLAY COUNTY,190724.4,190724.4,190724.4,190724.4,190724.4,192476.78,0,0,0,0,30.089579,-81.700455,Resident
333743,FL,CLAY COUNTY,0,79520.76,0,0,79520.76,86854.48,0,0,0,0,30.063236,-81.707703,Residential,Wood,CR
172534,FL,CLAY COUNTY,0,254281.5,0,254281.5,254281.5,246144.49,0,0,0,0,30.060614,-81.702675,Residential,Wood,CR
785275,FL,CLAY COUNTY,0,515035.62,0,0,515035.62,884419.17,0,0,0,0,30.063236,-81.707703,Residential,Masonry,CR
995932,FL,CLAY COUNTY,0,19260000,0,0,19260000,20610000,0,0,0,0,30.102226,-81.713882,Commercial,Reinforced Concrete
```

	A	B	C	D	E	F	G	H	I	J	K
1	policyID	statecode	county	eq_site_lir	hu_site_lir	fl_site_lim	fr_site_lim	tiv_2011	tiv_2012	eq_site_de	hu_site_de
2	119736	FL	CLAY COU	498960	498960	498960	498960	498960	792148.9	0	9979.2
3	448094	FL	CLAY COU	1322376	1322376	1322376	1322376	1322376	1438164	0	0
4	206893	FL	CLAY COU	190724.4	190724.4	190724.4	190724.4	190724.4	192476.8	0	0
5	333743	FL	CLAY COU	0	79520.76	0	0	79520.76	86854.48	0	0
6	172534	FL	CLAY COU	0	254281.5	0	254281.5	254281.5	246144.5	0	0
7	785275	FL	CLAY COU	0	515035.6	0	0	515035.6	884419.2	0	0
8	995932	FL	CLAY COU	0	19260000	0	0	19260000	20610000	0	0
9	223488	FL	CLAY COU	328500	328500	328500	328500	328500	348374.3	0	16425
10	433512	FL	CLAY COU	315000	315000	315000	315000	315000	265821.6	0	15750
11	142071	FL	CLAY COU	705600	705600	705600	705600	705600	1010843	14112	35280

Data Formats

XML – eXtensible Markup Language

- **Extensible** - Tags and document structure defined by the author.
- **Markup Language** – Annotation is syntactically distinguishable from the text/content.
- Human readable and machine readable

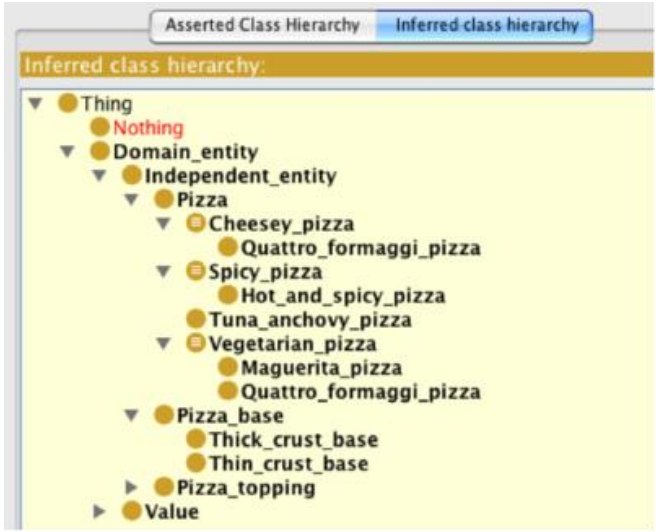
```
<menu>
        <item updatedDate="05/10/17">
                <name>Burger</name>
                <price>£4.00</price>
                <description>A beef burger with a choice of salad.</description>
                <calories>500</calories>
        </item>
        <item updatedDate="03/10/17">
                <name>Chips</name>
                <price>£2.00</price>
                <description>Chips with a choice of sauce.</description>
                <calories>300</calories>
        </item>
</menu>
```

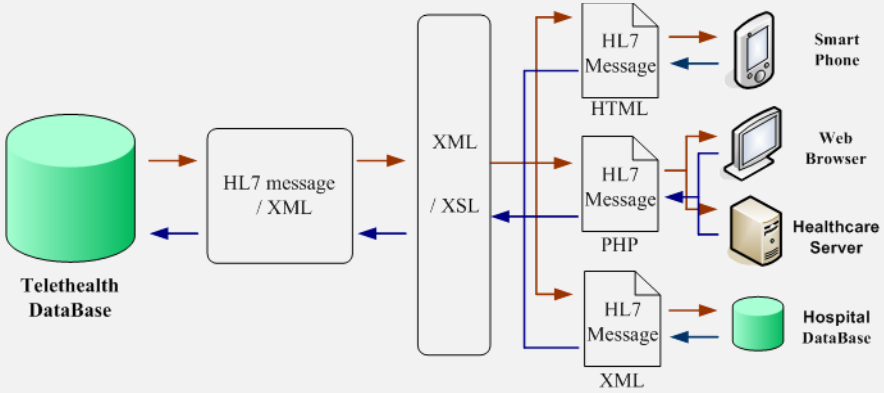
Ulster University

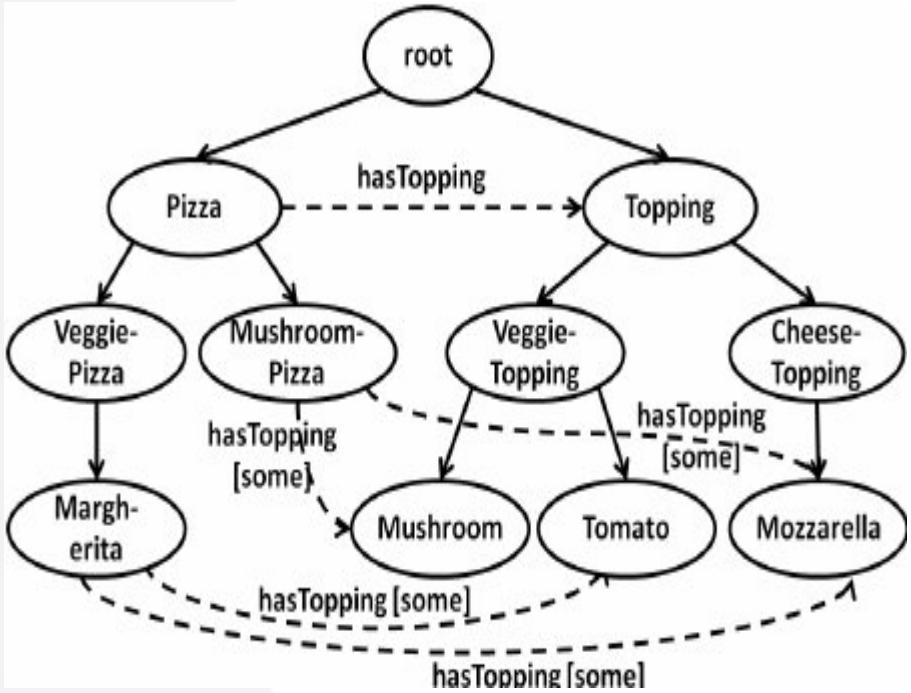
XML

Common Example

1. Semantic web→ Ontologies used
2. HL7 messages



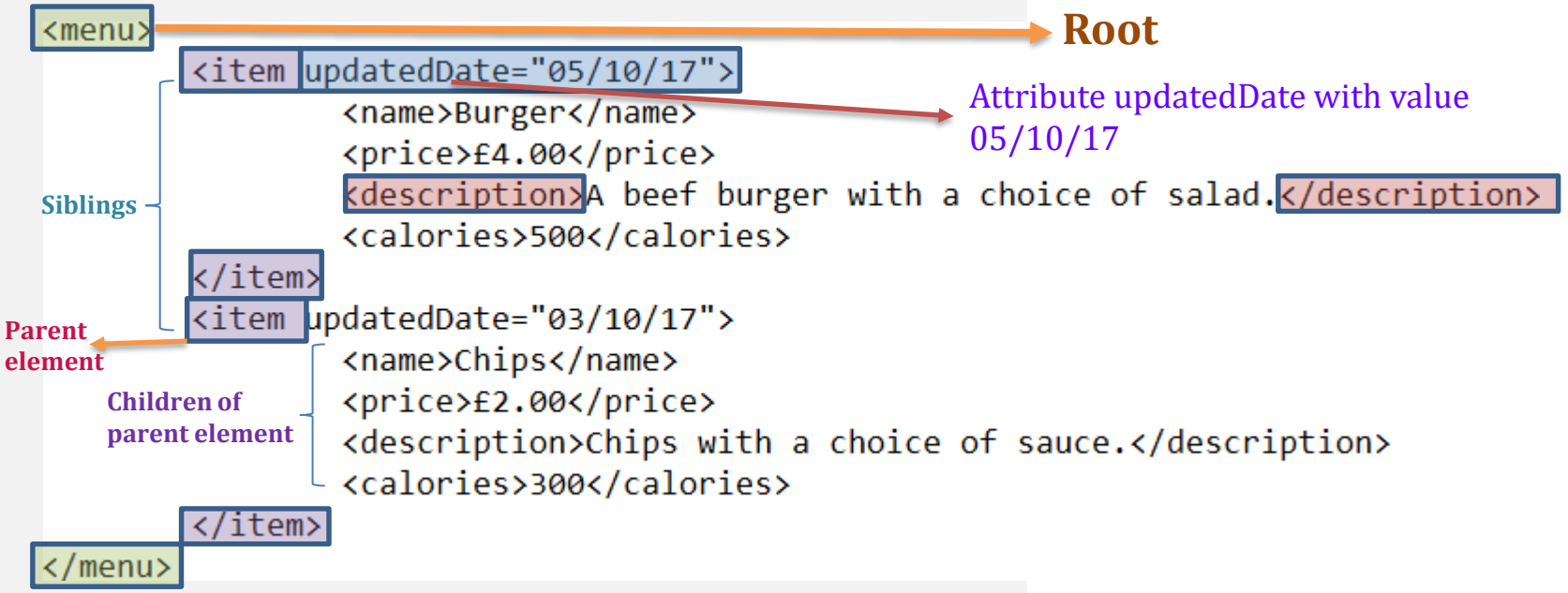




Ulster University

Data Formats

XML – eXtensible Markup Language



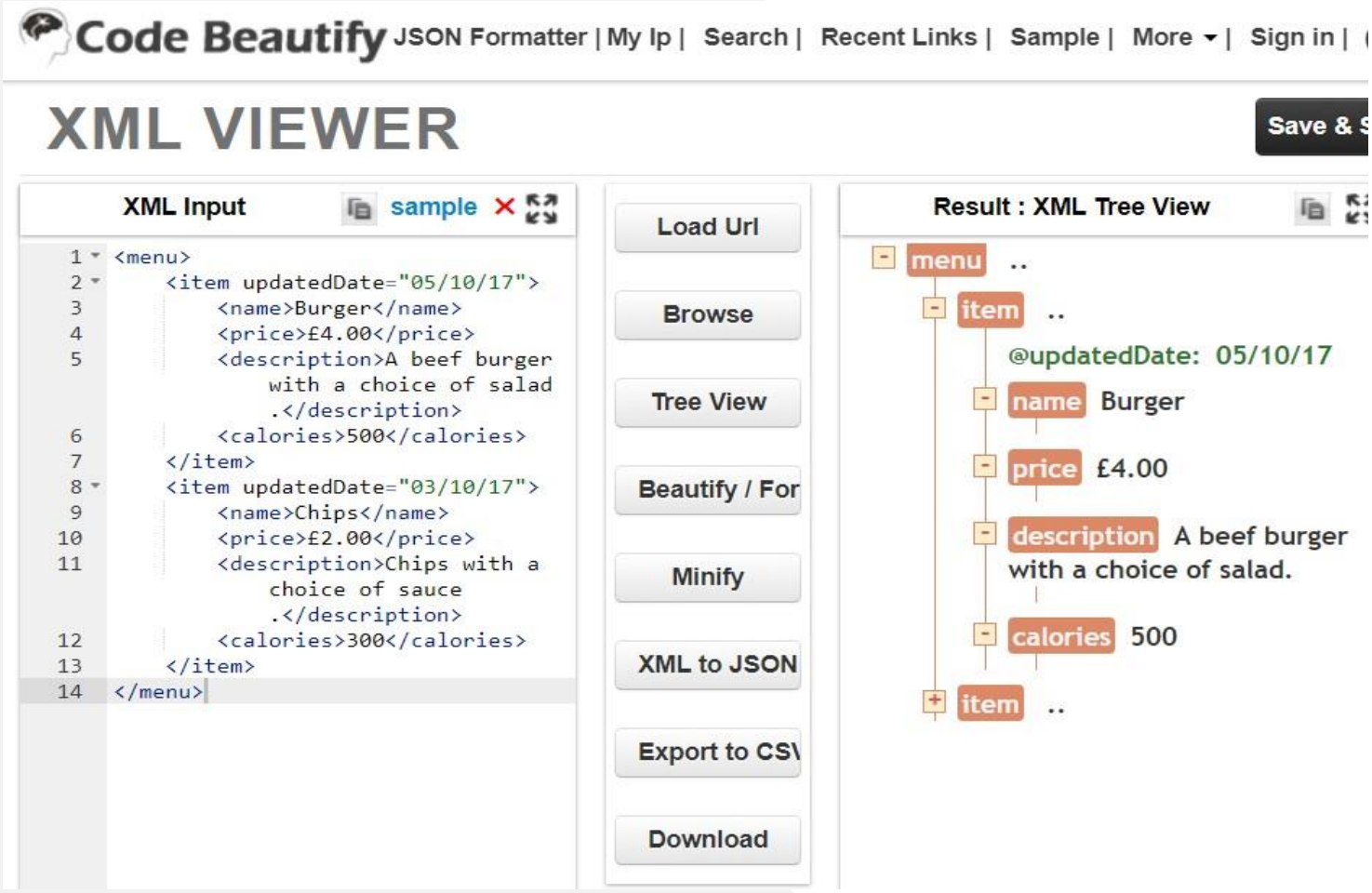
- A tree structure with:
  - A Root Element, Child Elements, Siblings and Parent Elements
  - Attributes

Ulster University

Data Formats

XML – eXtensible Markup Language

- http://codebeautify.org/xmlviewer



Ulster University

Data Formats

JSON– JavaScript Object Notation

- Lightweight data format
- Human and Machine-readable
- **Name: Value** pairs
- Objects held in curly brackets {}
- Arrays held in Square Brackets [ ]
- Syntax for storing and exchanging data

```
{"menu": [
                {
                        "name": "Burger",
                        "price": "£4.00",
                        "description": "A beef burger with a choice of salad.",
                        "calories": 500,
                        "updatedDate": "05/10/17"
                },
                {
                        "name": "Chips",
                        "price": "£2.00",
                        "description": "Chips with a choice of sauce.",
                        "calories": 300,
                        "updatedDate": "03/10/17"
                }
]}
```

Ulster University

Data Formats

JSON

- A syntax for storing and exchanging data
- It is a text
- Written with JavaScript object notation
- In client/server communication JSON objects are passed

Ulster University

Data Formats

Why use JSON?

- JSON format is text only→
  - ✓ Easy to send to and from a server
  - ✓ Used as a data format by any programming language
- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

  JSON.parse()

Ulster University

Data Formats

JSON

Syntax Rules: Derived from JavaScript object notation

- Data is in name(key): value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

```
myJson={ "name":"John" }
```

JSON Object

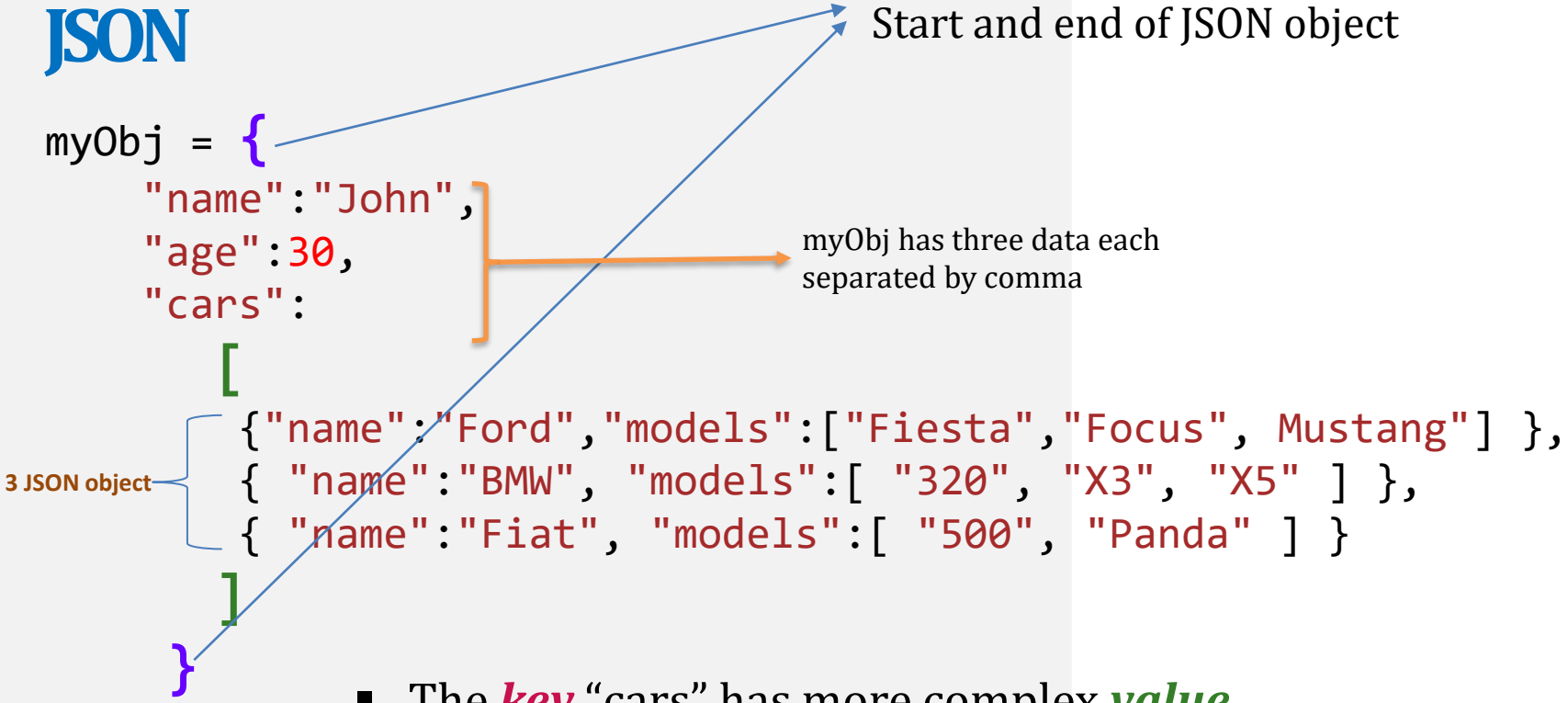
key : value

Note: *keys* must be strings, written with double quotes

Ulster University

Data Formats

JSON




```
myObj = {
    "name":"John",
    "age":30,
    "cars":
      [
       {"name":"Ford","models":["Fiesta","Focus", Mustang"] },
       { "name":"BMW", "models":[ "320", "X3", "X5" ] },
       { "name":"Fiat", "models":[ "500", "Panda" ] }
      ]
}
```

- The ***key*** "cars" has more complex ***value***
- ***value*** is ***array***
- ***array*** has three **objects**
- Each **object** has data "name" and "**models**"

"**models**" key has array of string values

Ulster University

Data Formats

JSON

- http://jsonviewer.stack.hu/

Ulster University

Data Storage Formats

Databases

- **Relational Databases**
  - Use a series of unique identifiers to match datasets
  - Data contained within tables linked by keys
  - Type of relationship is defined
  - Queried using SQL (Structured Query Language)
  - E.g. MySQL
- **Non-Relational Databases (NoSQL)**
  - Data stored in a flat format, usually JSON
  - MongoDB – One of the most popular NoSQL DB frameworks
- **Ontologies, Time Series, and many more!**

Ulster University

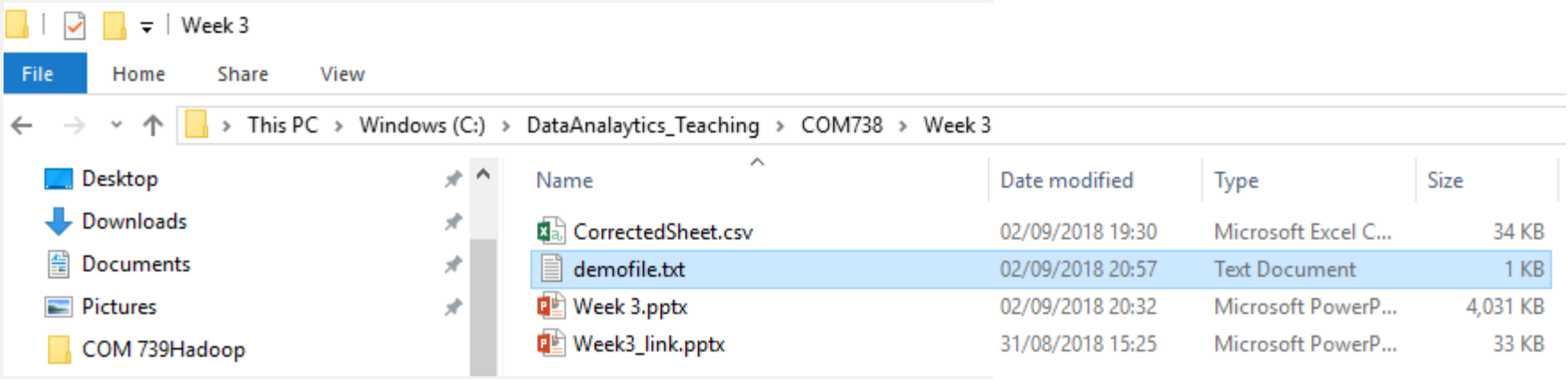
Ulster University

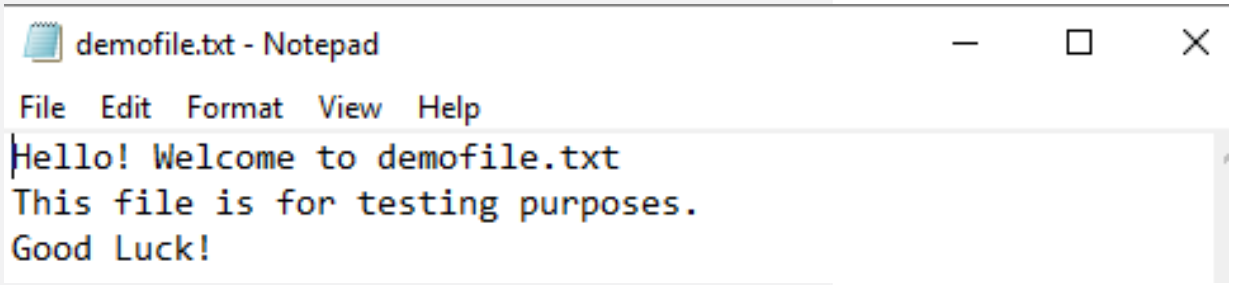
Reading Files in Python

ulster.ac.uk

Reading Files in Python

Let's assume you have a sample text file you wish to process





Ulster University

Reading Files in Python

- Interact with this file by creating a file object, using the open function
- **Note:**
  - Arguments (r, w, a, r+) are used to define the mode by which the file is opened
  - The *close* function must be called after interaction is complete so no limits are reached

```
#r = read. - Opens the file in read-only mode
file = open('samplefile.txt', 'r')

#w = write. Will destroy the file if it already exists!
file = open('samplefile.txt', 'w')

#a = append. Facilitates adding to the end of the file.
file = open('samplefile.txt','a')

#Always remember to close a file once finished with it.
file.close()
```

  - 'r+' mode will open a file in read and write mode

Ulster University

Reading Files in Python

Example:

```
1  myFile= open("demofile.txt","r")
2  print(myFile.read())
```

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

- The *myFile* **object** is an iterator which returns the next line in the file until the end is reached

Ulster University

Reading Files in Python

What could go wrong?

```
file = open('samplefile.txt','r')
for line in file:
    #Some more complex processing which
    #may result in an error
file.close()
```

Ulster University

Reading Files in Python

- **What could go wrong?**

```
file = open('samplefile.txt','r')
for line in file:
    #Some more complex processing which
    #may result in an error
file.close()
```

- A **try...finally** clause could handle this
- Python offers a convenient way of doing this

Ulster University

Reading Files in Python

- The *with* statement automatically closes a file upon exiting the code block, even if an exception occurs
- The basic structure is as follows:

```
with open('samplefile.txt','r') as file:
    #Extract the data from the file

#Process the data
```

- As before, use a *for* loop to iterate through each line in the file:

```
with open('samplefile.txt','r') as file:
    for line in file:
        print(line)
```

Ulster University

Python Modules

- A **module** is a Python file that consists of Python code
- Modules can define **functions**, **classes**, and **variables** that can be referenced in other Python files
- They are imported using the ***import*** statement:

```
import math
```

- Include any **import statements** at the top of your file or notebook
- Once a module is imported, objects within the module can be accessed and used

Ulster University

Python Modules

More on **Modules:**

- Functions/Statements/Variables within a module can be accessed using the syntax:
  - **<ModuleName>.<ObjectName>**

```
import math

print(math.sqrt(36))
```

```
6.0
```

```
import math

print(math.)
```

```
math.acos
math.acosh
math.asin
math.asinh
math.atan
math.atan2
math.atanh
math.ceil
math.copysign
math.cos
```

Ulster University

Python Modules

More on **Modules:**

- A specific function in a module can be imported and called as follows:

```
from math import sqrt

print(sqrt(36))
```

```
6.0
```

Directly called

- Modules can be **aliased**, i.e. referred to as a different name as follows:

```
import math as m

print(m.sqrt(36))
```

```
6.0
```

Ulster University

Reading CSV Files in Python

- CSV Files can be read with the use of a ***csv.reader***
- This must be imported from the ***csv* module**

```
import csv
```

Ulster University

Reading CSV Files in Python

A *csv.reader* is used very similarly to our previous file object:

```
import csv

with open("PracticalFiles/csv_modules.csv") as csvfile:
    readcsv = csv.reader(csvfile,delimiter=',')
    for row in readcsv:
        print("Full row: ",row)
        print("Row[0]: ", row[0])
        print("Row[1]: ", row[1])
```

Complete full row value is in row variable for each iteration

Get each value per column in a particular row

```
module_name,module_code
Data Science Foundations,COM738
Business Intelligence,COM735
Databases and Cloud Computing,COM739
Masters Project (Research),COM865
```

Ulster University

Reading CSV Files in Python

The output:

```
    for row in readcsv:
        print("Full row: ",row)
        print("Row[0]: ", row[0])
        print("Row[1]: ", row[1])
```

```
Full row:  ['module_name', 'module_code']
Row[0]:  module_name
Row[1]:  module_code
Full row:  ['Data Science Foundations', 'COM738']
Row[0]:  Data Science Foundations
Row[1]:  COM738
Full row:  ['Business Intelligence', 'COM735']
Row[0]:  Business Intelligence
Row[1]:  COM735
Full row:  ['Databases and Cloud Computing', 'COM739']
Row[0]:  Databases and Cloud Computing
Row[1]:  COM739
Full row:  ['Masters Project (Research)', 'COM865']
Row[0]:  Masters Project (Research)
Row[1]:  COM865
```

Ulster University

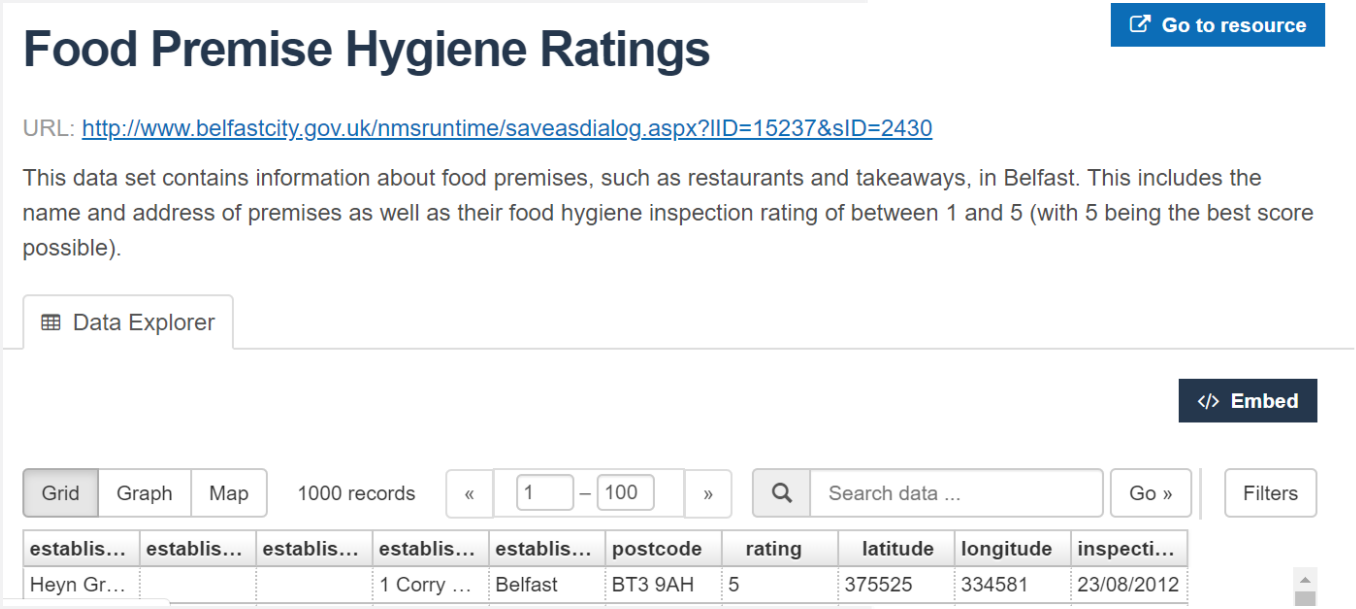
Reading CSV Files in Python

An example with more complex data

Source:

- OpenDataNI – Food Premise Hygiene Ratings

https://www.opendatani.gov.uk/dataset/food-premise-hygiene-ratings/resource/3d998bd3-ecbe-4087-a653-ea11448ea53f



Food Premise Hygiene Ratings

Go to resource

URL: http://www.belfastcity.gov.uk/nmsruntime/saveasdialog.aspx?lID=15237&sID=2430

This data set contains information about food premises, such as restaurants and takeaways, in Belfast. This includes the name and address of premises as well as their food hygiene inspection rating of between 1 and 5 (with 5 being the best score possible).

Data Explorer

Embed

Grid | Graph | Map | 1000 records | « | 1 – 100 | » | Search data ... | Go » | Filters

establis...	establis...	establis...	establis...	establis...	postcode	rating	latitude	longitude	inspecti...
Heyn Gr...			1 Corry ...	Belfast	BT3 9AH	5	375525	334581	23/08/2012

Ulster University

Reading CSV Files in Python

The csv module's ***DictReader*** can provide each column as a ***dict*** *with* the headers as keys:

```
1  import csv
2  with open('foodhygienedata.csv', 'r') as file:
3      input_file = csv.DictReader(file,delimiter=',')
4      for column in input_file:
5              name =column["establishmentname"]
6              print(name)
```

Getting all the rows of column named establishmentname

```
Heyn Group
Rosemary Lunch Club
John Ross & Co Auctioneers
The Maverick/Boom Box
Maverick
Windsor Recreation & Social Cl
```

- Header as *key* and then you can get complete value in that column

Ulster University

Ulster University

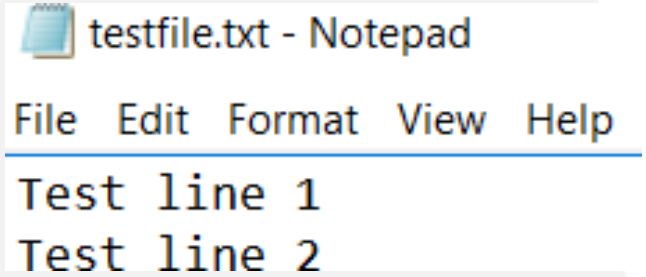
Writing to Files in Python

ulster.ac.uk

Writing to Plaintext Files in Python

- We may wish to save any results for future reference
- Writing **plaintext** files is a very straightforward
- As with reading, we first create a ***file* object**
- The file is opened in one of the following modes: **w, r+, a**
- We then use the ***write* function** to output the desired values
- Finally, we must **close** the file

```
with open("testfile.txt","w") as file:
    file.write("Test line 1 \n")
    file.write("Test line 2")
```



Ulster University

Writing to CSV Files in Python

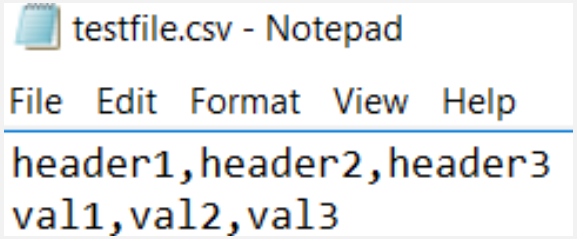
Writing to CSV files requires a **CSV Writer**

- This will automatically convert provided data into delimited strings and output to the desired file
- Again, this must be imported from the csv module

```
import csv

headers=["header1","header2","header3"]
values=["val1","val2","val3"]


with open("testfile.csv","w",newline="") as file:
    writer = csv.writer(file,delimiter=',')
    writer.writerow(headers)
    writer.writerow(values)
```



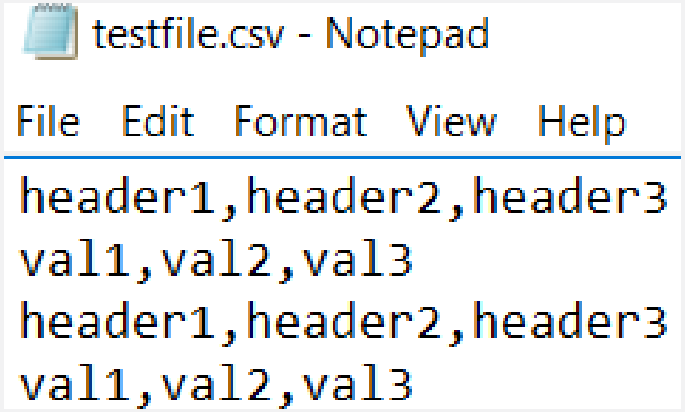
	A	B	C
1	header1	header2	header3
2	val1	val2	val3

Ulster University

Writing to CSV Files in Python

Experimenting with "append" mode(a)

- Specifying mode "w" will overwrite the file each time it is opened
- Append mode (a) will append data to the end of an existing file, e.g.




```python
import csv

headers=["header1","header2","header3"]
values=["val1","val2","val3"]


with open("testfile.csv","a",newline="") as file:
    writer = csv.writer(file,delimiter=',')
    writer.writerow(headers)
    writer.writerow(values)

with open("testfile.csv","a",newline="") as file:
    writer = csv.writer(file,delimiter=',')
    writer.writerow(headers)
    writer.writerow(values)
```

Ulster University

Ulster University

Task

Task

Create a **CSV File,** that contain a
list of your Semester 1 modules with the following information:

- Module Name
- Module Code

Save these files for use in the remainder of the class

Ulster University

Task Solution

CSV:

```
module_name,module_code
Data Science Foundations,COM738
Business Intelligence,COM735
Databases and Cloud Computing,COM739
Masters Project (Research),COM865
```

Ulster University

Task

- Create a plaintext file using notepad, containing a few sentences on separate lines
- Save this file in the same directory as your Jupyter Notebook
- Create a Python function that will output something similar to the following:

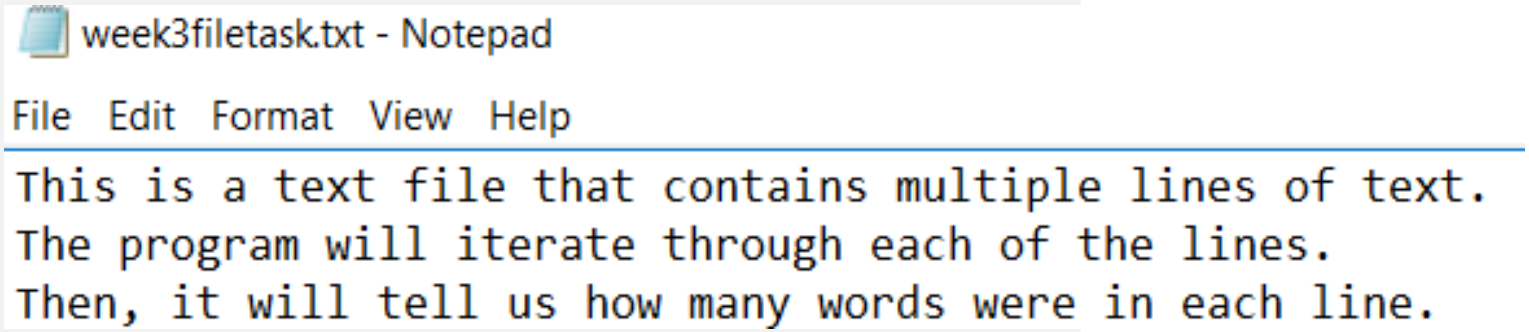
```
11 words in: This is a text file that contains multiple lines of text.

9 words in: The program will iterate through each of the lines.

12 words in: Then, it will tell us how many words were in each line.
```

Ulster University

Task Solution




```
with open('week3filetask.txt','r') as file:
    for line in file:
        words = line.split(' ')
        print("{0} words in: {1}".format(len(words),line))
```

Ulster University

Task

- Access and download the ***CSV VERSION*** of the Belfast Food Premise Hygiene Ratings dataset
- Save it to the same directory as your Jupyter Notebook
  1. Print a list containing all field names in the header
  2. Print a list of all postcodes
  3. Print a list of all establishment names that do not have a recorded postcode.
  4. Print a list of all establishment names that are missing any item of information
- Hint – Use the "any" function – Stack Overflow has many examples

Ulster University

Task - Solutions

- Print a list containing all field names in the header:

```
with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=",")
    print(reader.fieldnames)
```

- Print a list of all postcodes:

```
import csv

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=',')
    for row in reader:
        postcode = row["postcode"]
        print(postcode)
```

Ulster University

Task - Solutions

- Print a list of all establishment names that do not have a recorded postcode

```
import csv

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=',')
    for row in reader:
        postcode = row["postcode"]
        if not postcode:
            establishment = row["establishmentname"]
            print(establishment)
```

- Note: "not" returns True for 0, "", None, and False

Ulster University

Task - Solutions

- Print a list of all establishment names that are missing any item of information
  - Hint: use the "any" function
- The "**any**" function:
  - Returns **True** on the first encounter of a condition evaluating to True, else returns False
- Similarly, the "**all**" function:
  - Returns **False** on the first encounter of a condition evaluating to False, else returns True

Ulster University

Task - Solutions

- Print a list of all establishment names that are missing any item of information

```
import csv

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=',')
    for row in reader:
        if any(row[key] in (None,"") for key in row):
            print(row["establishmentname"])
```

```
Heyn Group
Rosemary Lunch Club
John Ross & Co Auctioneers
The Maverick/Boom Box
```

- It appears that all records in this dataset have at least one field missing!

Ulster University

Task - Solutions

- We can amend the first record in the dataset manually to populate all fields and ensure our approach works

```
1  establishmentname,establishmentaddressline1,establishmentaddressline2,estab
   ,latitude,longitude,inspectiondateCRLF
2  Heyn Grou,,,1 Corry Place,Belfast,BT3 9AH,5,375525,334581,23/08/2012CRLF
```

```
1  establishmentname,establishmentaddressline1,establishmen
   ,latitude,longitude,inspectiondateCRLF
2  Heyn Group,Authur Building,Room 1,1 Corry Place,Belfast,
```

Ulster University

Task - Solutions

- If we re-run our previous code, the Heyn Group is no longer printed, as all fields have values

```
import csv

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=',')
    for row in reader:
        if any(row[key] in (None,"") for key in row):
            print(row["establishmentname"])
```

```
Rosemary Lunch Club
John Ross & Co Auctioneers
The Maverick/Boom Box
```

Ulster University

Task

1. Modify the “Heyn Group” record in the dataset so that all fields are populated
2. Use the “all” function to print a list of all establishment names in records that have no missing fields (only the Heyn Group will be printed)

Ulster University

Task

Write a function which will:

- Read in the establishment names and inspection dates listed in the Food Hygiene Dataset
- For each record, calculate the value "DaysSinceInspection". This value contains the number of days since the date specified in the "inspectiondate" field
- Note: You will need to use the string "**split**" method on each date from the CSV file, and then create a new **Date** object from these string components. You will then need to subtract this new Date object from today's date to find the difference. Do some Googling around this area! You will also need to deal appropriately with records that have no date value (use an if statement)
- Create a new CSV file which contains a collection of establishment names, inspection dates, and days since inspection.

Ulster University

Task - Solutions

Step 1 - Read in the establishment names and inspection dates listed in the Food Hygiene Dataset

```
import csv
from datetime import date

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file, delimiter=',')
    enames = []
    idates = []
    dsi = []
    for row in reader:
        enames.append(row["establishmentname"])
        idates.append(row["inspectiondate"])
        current_idate = row["inspectiondate"]
```

Ulster University

Task - Solution

Step 2 – Check if there is a date for each row. If there is:

- Split the date string into separate components (day, month, year)
- Calculate the number of days since the inspection date

```
if current_idate:
    dtcomponents = current_idate.split("/")
    dt = date(int(dtcomponents[2]), int(dtcomponents[1]),int(dtcomponents[0]))
    dtdifference = date.today() - dt

    dsi.append(dtdifference.days)
else:
    dsi.append("")
```

Ulster University

Task - Solution

Step 3 – Write the values to a new file

```
with open("PracticalFiles/foodhygienedata_DAYS.csv", "w", newline="") as file:
    writer = csv.writer(file,delimiter=",")
    headers = ["establishmentname","inspectiondate","DaysSinceInspection"]
    writer.writerow(headers)

    for ename,idate,numdays in zip(enames,idates,dsi):
          writer.writerow([ename,idate,numdays])
```

Ulster University

Task– Complete Solution

```
import csv
from datetime import date

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file, delimiter=',')
    enames = []
    idates = []
    dsi = []
    for row in reader:
        enames.append(row["establishmentname"])
        idates.append(row["inspectiondate"])
        current_idate = row["inspectiondate"]

        if current_idate:
            dtcomponents = current_idate.split("/")
            dt = date(int(dtcomponents[2]), int(dtcomponents[1]),int(dtcomponents[0]))
            dtdifference = date.today() - dt

            dsi.append(dtdifference.days)
        else:
            dsi.append("")

with open("PracticalFiles/foodhygienedata_DAYS.csv", "w", newline="") as file:
    writer = csv.writer(file,delimiter=",")
    headers = ["establishmentname","inspectiondate","DaysSinceInspection"]
    writer.writerow(headers)

    for ename,idate,numdays in zip(enames,idates,dsi):
          writer.writerow([ename,idate,numdays])
```

Ulster University

Task - Solutions

- Use the "all" function to print a list of all establishment names in records that have no missing fields

```
import csv

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=',')
    for row in reader:
        if all(row[key] for key in row):
            print(row["establishmentname"])
```

```
Heyn Group
```

- row[key] will evaluate to True if a value is present, which is not False, None, 0, " " or an empty container.

Ulster University

Task

- Generate the following metrics from the dataset:
  - A count of how many establishments received each rating: 1, 2, 3, 4 and 5, and how many ratings were omitted
  - Output should look similar to the following:

```
Amount scoring 1:  28
Amount scoring 2:  61
Amount scoring 3:  391
Amount scoring 4:  820
Amount scoring 5:  1786
Amount missing:  82
```

Time: **30 minutes**

Ulster University

Task - Solution

A count of how many establishments received each rating: 1, 2, 3, 4 and 5, and how many ratings were omitted:

```
import csv

with open("PracticalFiles/foodhygienedata.csv") as file:
    reader = csv.DictReader(file,delimiter=',')
    ratings = [0,0,0,0,0]
    omittedRatings = 0
    for row in reader:
        if row["rating"]=="5":
            ratings[4]+=1
        if row["rating"]=="2":
            ratings[1]+=1
        if row["rating"]=="3":
            ratings[2]+=1
        if row["rating"]=="4":
            ratings[3]+=1
        if row["rating"]=="1":
            ratings[0]+=1
        if row["rating"] in (None,""):
            omittedRatings+=1
    print("Amount scoring 1: ", ratings[0])
    print("Amount scoring 2: ", ratings[1])
    print("Amount scoring 3: ", ratings[2])
    print("Amount scoring 4: ", ratings[3])
    print("Amount scoring 5: ", ratings[4])
    print("Amount missing: ", omittedRatings)
```

Ulster University

Cleaning Data – A Brief Introduction

- Replacing abbreviated headers
- Formatting Data
- Finding Outliers & Bad Data
  - Careful consideration – don't want to manipulate results.
  - Identify and handle **missing values** appropriately
  - Check **data types** (e.g. a string found where an int expected)
  - Identify and handle **duplicate values** – an error in one dataset, or perhaps when merging multiple datasets

Ulster University

Cleaning Data – A Brief Introduction

- The Food Hygiene Dataset has illustrated how a dataset may not be perfect (though it very close!)
- The process of **Data Cleaning** is an important part of the data wrangling process
- Python allows us to build cleaning functions around patterns, eliminating repetitive work
- Once cleaning is complete, we can analyse the data, as we have explored today

Ulster University

Final Task

- Take some time to explore openly available online datasets from the sources recommended in the first part of today's class, then:
  - Locate a CSV dataset. Download and save.
  - Read/Print in the entire contents
  - Experiment with processing the file, for example:
    - Identifying missing records
    - Identifying duplicates
    - Generate useful metrics about the file.
  - Time: **Remainder of class**

Ulster University