SIM Global Education

Topic 3
Data Structures

Learning Outcomes

After completing this topic and the recommended reading, you should be able to:

- Explain the difference between data structures and select appropriate data structures for particular examples of data.
- Write Python programs that can process and analyse text data.
- Implement linguistic analysis algorithms.

1. Data Representation

Representing Numbers

- Binary notation
  - Uses bits to represent a number in base two
- The **binary** system is based on powers of two
  - Base 2
  - The state of on/off in electronics
- The traditional **decimal** system is based on powers of ten
  - Base 10
  - Make sense to human
- The **hexadecimal** system is based on powers of sixteen
  - Base 16
  - A shorthand notation for long bit patterns, in group of four binary digits

Representing Text

- Human understands languages/characters
- Computer understands "1"s & "0"s
- Each character is assigned a unique bit pattern
  - Printable characters
    - Letters; punctuations; numbers; symbols (e.g. @)
  - Control characters
    - Controlling display (e.g. tab); communication (e.g. beep)
- **Character coding** converts characters into bit patterns that computer can understand
  - ASCII (7-bits) & Extended ASCII (8-bits)

    - Unicode (16-bits)

Representing Image

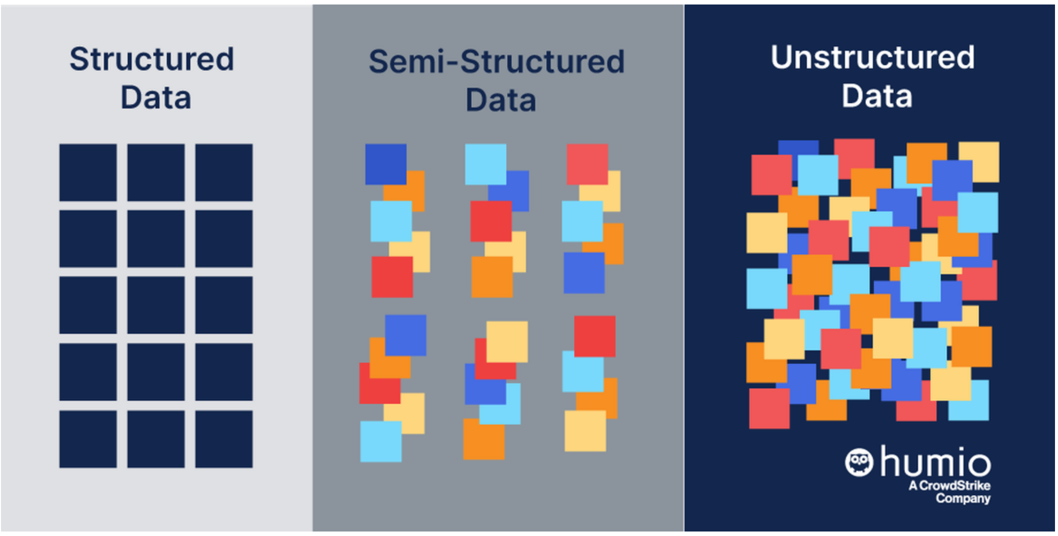
- One means of representing an image is to interpret the image as a collection of dots
    - **Pixel** (Picture Element)
- Each pixel is encoded with information that represent the characteristics of an image, such as, colours, brightness, etc.
- The collection of these encoded pixels is referred to as **Bit Map**
    - Stores images in a set of bits
    - Binary image
        - Black / White: 1 bit
    - Gray image
        - Varying shades: 8 bits (0-255)
    - Colour image
        - **RGB** (red-green-blue): 24 bits (3 x 8 bits)

Representing Sound

- Sound is an acoustic wave, a simple wave can be characterized by amplitude and frequency
    - The larger the amplitude the louder the sound
    - The higher the frequency the higher the pitch
- Sampling techniques
    - Used for high quality recordings
    - Records actual audio
- Analog sound signal need to be converted into a digital format, to be stored and transmitted within computer
    - Analog waves: Nature

- Digital pulses: Computer

<u>2. Data Categorisation</u>



[Source: *https://www.humio.com/whats-new/blog/structured-logging-explained/*]

Structured Data

- Resides in predefined formats and models.
- Generally tabular data that is represented by columns (fields) and rows (records).
- Examples: relational databases

timestamp	latitude	longitude	altitude	distance	heart_rate	speed
2013-06-01 18:40:29	50.81381	-1.712606	80.20001	1805.94	133	4.060059
2013-06-01 18:40:30	50.81383	-1.712649	80.00000	1810.00	133	4.550049
2013-06-01 18:40:31	50.81385	-1.712700	79.79999	1814.55	133	2.979981
2013-06-01 18:40:32	50.81387	-1.712734	79.79999	1817.53	133	2.969971
2013-06-01 18:40:33	50.81388	-1.712777	79.59998	1820.50	133	3.650024
2013-06-01 18:40:34	50.81389	-1.712826	79.59998	1824.15	133	3.229980
2013-06-01 18:40:35	50.81391	-1.712862	79.40002	1827.38	133	4.650024
2013-06-01 18:40:36	50.81393	-1.712911	79.40002	1832.03	133	4.149902
2013-06-01 18:40:37	50.81395	-1.712963	79.20001	1836.18	133	2.000000
2013-06-01 18:40:38	50.81395	-1.712994	79.20001	1838.18	133	4.210083
2013-06-01 18:40:39	50.81396	-1.713053	79.00000	1842.39	133	5.189941

Unstructured Data

- Information that is text-heavy but may contain data such as numbers, dates, and facts.
- Stored in its natural format until it's extracted for analysis.
- Examples: videos; audio; and binary data files

Semi-structured Data

- Information that doesn't consist of structured data but still has some structure to it.
- A mix of both structured and unstructured data.
- Example: documents held in JSON format; XML files

3. Text File Formats

Plain Text

- Represents only characters of readable material but not its graphical representation nor other objects.
- May include *whitespace* characters that affect simple arrangement of text, such as spaces, line breaks, or tabulation characters.
- Extension: **.txt**

nile.txt

```
1120 1160 963 1210 1160 1160 813 1230 1370 1140 995 935 1110 994 1020 960 1180 799 958
1140 1100 1210 1150 1250 1260 1220 1030 1100 774 840 874 694 940 833 701 916 692 1020 1050
969 831 726 456 824 702 1120 1100 832 764 821 768 845 864 862 698 845 744 796 1040 759 781
865 845 944 984 897 822 1010 771 676 649 846 812 742 801 1040 860 874 848 890 744 749 838
1050 918 986 797 923 975 815 1020 906 901 1170 912 746 919 718 714 740
```

sunset-salvo.txt

```
The combination of some data and an aching desire for an answer does not ensure that a
reasonable answer can be extracted from a given body of data
```

Delimiter-Separated Values

- Stores two-dimensional arrays of data by separating the values in each row with specific delimiter characters, such as tabs, or commas
- Extension: **.tsv**; **.csv**

running_dat.tsv

```
"timestamp"	"latitude"	"longitude"	"altitude"	"distance"	"heart_rate"	"speed"
2013-06-01 18:40:29	50.813805	-1.7126063	80.2000122	1805.9399512	133	4.06005860000005
2013-06-01 18:40:30	50.8138298	-1.7126487	80	1810.0000098	133	4.55004880000001
2013-06-01 18:40:31	50.8138543	-1.7127005	79.7999878	1814.5500586	133	2.97998050000001
2013-06-01 18:40:32	50.8138709	-1.7127338	79.7999878	1817.5300391	133	2.96997069999998
2013-06-01 18:40:33	50.8138757	-1.7127769	79.5999756	1820.5000098	133	3.65002439999989
2013-06-01 18:40:34	50.8138862	-1.712826	79.5999756	1824.1500342	133	3.22998040000016
2013-06-01 18:40:35	50.8139051	-1.7128616	79.4000244	1827.3800146	133	4.65002449999997
2013-06-01 18:40:36	50.8139326	-1.7129115	79.4000244	1832.0300391	133	4.14990229999989
2013-06-01 18:40:37	50.8139517	-1.7129626	79.2000122	1836.1799414	133	2
2013-06-01 18:40:38	50.8139537	-1.7129945	79.2000122	1838.1799414	133	4.21008299999994
2013-06-01 18:40:39	50.8139622	-1.7130533	79	1842.3900244	133	5.18994140000018
```

running_dat.csv

```
"timestamp","latitude","longitude","altitude","distance","heart_rate","speed"
2013-06-01 18:40:29,50.813805,-1.7126063,80.2000122,1805.9399512,133,4.06005860000005
2013-06-01 18:40:30,50.8138298,-1.7126487,80,1810.0000098,133,4.55004880000001
2013-06-01 18:40:31,50.8138543,-1.7127005,79.7999878,1814.5500586,133,2.97998050000001
2013-06-01 18:40:32,50.8138709,-1.7127338,79.7999878,1817.5300391,133,2.96997069999998
2013-06-01 18:40:33,50.8138757,-1.7127769,79.5999756,1820.5000098,133,3.65002439999989
2013-06-01 18:40:34,50.8138862,-1.712826,79.5999756,1824.1500342,133,3.22998040000016
2013-06-01 18:40:35,50.8139051,-1.7128616,79.4000244,1827.3800146,133,4.65002449999997
2013-06-01 18:40:36,50.8139326,-1.7129115,79.4000244,1832.0300391,133,4.14990229999989
2013-06-01 18:40:37,50.8139517,-1.7129626,79.2000122,1836.1799414,133,2
2013-06-01 18:40:38,50.8139537,-1.7129945,79.2000122,1838.1799414,133,4.21008299999994
2013-06-01 18:40:39,50.8139622,-1.7130533,79,1842.3900244,133,5.18994140000018
```

XML (eXtensible Markup Language)

- Defines a set of rules for encoding documents (structured and semi-structured) in a format that is both human-readable and machine-readable
- Simple and very flexible text format derived from SGML (Standard Generalized Markup Language)
- Great format for storing hierarchical data
- Syntax:

  <markup> content </markup>

  <element>

  <child element> data </child element>

  </element>
- Extension: **<u>.xml</u>**

statisticians.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- A list of famous statisticians -->
<records>
  <statistician>
    <name>Ronald</name>
    <middle>Aylmer</middle>
    <surname>Fisher</surname>
    <dob>17/02/1890</dob>
    <wiki>https://en.wikipedia.org/wiki/Ronald_Fisher</wiki>
  </statistician>
  <statistician>
    <name>William</name>
    <middle>Sealy</middle>
    <surname>Gosset</surname>
    <dob>13/08/1876</dob>
    <wiki>https://en.wikipedia.org/wiki/William_Sealy_Gosset</wiki>
  </statistician>
  <statistician>
    <name>David</name>
    <middle>Roxbee</middle>
    <surname>Cox</surname>
    <dob>15/07/1924</dob>
    <wiki>https://en.wikipedia.org/wiki/David_Cox_(statistician)</wiki>
  </statistician>
  <statistician>
```

JSON (JavaScript Object Notation)

- Open standard file format and lightweight data interchange format, easy for humans to read and write, and easy for machines to parse and generate.

- Uses human-readable text to store and transmit data objects consisting of attribute-value pairs and arrays.
- Supports basic variable types, including strings, numbers, Booleans, null, arrays and objects.
- Syntax:

  Object: { … … }

  “key”: “value”

```
{
    "language": "Python",
    "release": 1991,
    "os": ["Linux", "macOS", "Windows"],
    "oo": true,
    "pastnames": null
}
```

- Extension: **<u>.json</u>**

statisticians.json

```
[
    {
        "name": "Ronald",
        "middle": "Aylmer",
        "surname": "Fisher",
        "dob": "17/02/1890",
        "wiki": "https://en.wikipedia.org/wiki/Ronald_Fisher"
    },
    {
        "name": "William",
        "middle": "Sealy",
        "surname": "Gosset",
        "dob": "13/08/1876",
        "wiki": "https://en.wikipedia.org/wiki/William_Sealy_Gosset"
    },
    {
        "name": "David",
        "middle": "Roxbee",
        "surname": "Cox",
        "dob": "15/07/1924",
        "wiki": "https://en.wikipedia.org/wiki/David_Cox_(statistician)"
    },
    {
        "name": "Thomas",
        "middle": null,
        "surname": "Bayes"
```

Spreadsheets

- Computer application for organisation, analysis, and storage of data in tabular form.
- Program operates on data (numeric, text, or formulas) entered in cells of a table.

- Example: Microsoft Excel; LibreOffice Calc; Apple Numbers
- Extension: **<u>.xlsx</u>**; **<u>.ods</u>**; **<u>.numbers</u>**

Sheet 1

running_dat

	A	B	C	D	E	F	G
1	timestamp	latitude	longitude	altitude	distance	heart_rate	speed
2	2013-06-01 18:40:29	50.813805	-1.7126063	80.2000122	1805.9399512	133	4.06005860000005
3	2013-06-01 18:40:30	50.8138298	-1.7126487	80	1810.0000098	133	4.55004880000001
4	2013-06-01 18:40:31	50.8138543	-1.7127005	79.7999878	1814.5500586	133	2.97998050000001
5	2013-06-01 18:40:32	50.8138709	-1.7127338	79.7999878	1817.5300391	133	2.96997069999998
6	2013-06-01 18:40:33	50.8138757	-1.7127769	79.5999756	1820.5000098	133	3.65002439999989

4. Reading Files in Python

Import Plain Text Files in Python

- Read data from a file
  - *text_file = open('Sentence.txt', 'r')*

```
Sentence.txt
All models are wrong, but some are useful
```

  - *lines = text_file.read()*
- Show variable content
  - *lines*

```
'All models are wrong, but some are useful'
```

- Close the file object
  - *text_file.close()*
- Tokenizing the words
  - *words = lines.split()* *# default: space " "*
  - *words*

```
['All', 'models', 'are', 'wrong,', 'but', 'some', 'are', 'useful']
```

Import/Export CSV Files in Python

- Read data as data frame a file
  - *import pandas as pd*
  - *df = pd.read_csv('titanic.csv')*

```
titanic.csv
survived,pclass,sex,age,sibsp,parch,fare,embarked,class,who,adult_male,deck,embark_town,alive,alone,
0,3,male,22,1,0,7.25,S,Third,man,TRUE,,Southampton,no,FALSE,
1,1,female,38,1,0,71.2833,C,First,woman,FALSE,C,Cherbourg,yes,FALSE,
1,3,female,26,0,0,7.925,S,Third,woman,FALSE,,Southampton,yes,TRUE,
1,1,female,35,1,0,53.1,S,First,woman,FALSE,C,Southampton,yes,FALSE,
0,3,male,35,0,0,8.05,S,Third,man,TRUE,,Southampton,no,TRUE,
0,3,male,,0,0,8.4583,Q,Third,man,TRUE,,Queenstown,no,TRUE,
0,1,male,54,0,0,51.8625,S,First,man,TRUE,E,Southampton,no,TRUE,
0,3,male,2,3,1,21.075,S,Third,child,FALSE,,Southampton,no,FALSE,
1,3,female,27,0,2,11.1333,S,Third,woman,FALSE,,Southampton,yes,FALSE,
1,2,female,14,1,0,30.0708,C,Second,child,FALSE,,Cherbourg,yes,FALSE,
1,3,female,4,1,1,16.7,S,Third,child,FALSE,G,Southampton,yes,FALSE,
```

- Show variable content
  - *df*

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

A view of the previous CSV file as a pandas data frame.

- Export
  - *df.to_csv(r'path', index = False, header = True)*

Import/Export JSON Files in Python

- Read data as data frame a file
  - *import pandas as pd*
  - *df = pd.read_json('example.json')*

example.json

```
{
        "Product":
                {"0":"Desktop Computer","1":"Tablet","2":"iPhone","3":"Laptop"},
        "Price":
                {"0":700,"1":250,"2":800,"3":1200}
}
```

- Show variable content
  - *df*

	Product	Price
0	Desktop Computer	700
1	Tablet	250
2	iPhone	800
3	Laptop	1200

- Export

- *df.to_json(r'path)*

Import/Export XML Files in Python

- Read data from file
  - *import xml.etree.ElementTree as et*
  - *tree = et.parse('example.xml')*

example.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
    <items>
        <item name="item1">item1 abc</item>
        <item name="item2">item2 abc</item>
    </items>
</data>
```

  - *root = tree.getroot()*
  - *print('Item #1 attribute: ', root[0][0].attrib)*

```
Item #1 attribute:  {'name': 'item1'}
```

  - *print('\Item #2 data: ', root[0][1].text)*

```
\Item #2 data:  item2 abc
```

4. Natural Language Toolkit (NLTK)

- The ***Natural Language Toolkit***, or more commonly ***NLTK***, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.
- It is a leading platform for building Python programs to work with human language data.
- https://www.nltk.org

Natural Language Processing

- ***Natural language processing*** is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyse large amounts of natural language data.
- It is about developing applications and services that can understand human languages.
- Examples:
    - Speech recognition
    - Speech translation
    - Understanding complete sentences
    - Understanding synonyms of matching words
    - Writing complete grammatically correct sentences and paragraphs

Installing/Importing NLTK Library

- *conda install -c conda-forge nltk*
- *pip install nltk*
- *import nltk*

- *nltk.download()*

Tokenizing

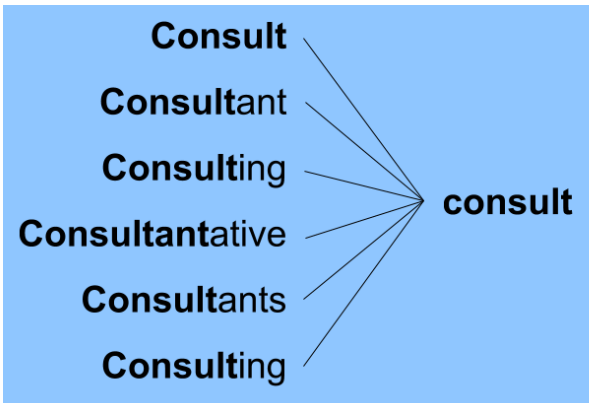
- NLTK contains a module called ***tokenize*** with a *word_tokenize()* method that will help us split a text into tokens
- *Punkt Sentence Tokenizer* divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences.
- Example:
    - *from nltk import word_tokenize*
    - *nltk.download("punkt")* *# punkt sentence tokenizer*
    - *word_tokenize(lines)*

    ```
    ['All', 'models', 'are', 'wrong', ',', 'but', 'some', 'are', 'useful']
    ```
    - *nltk.FreqDist(words)*

    ```
    FreqDist({'are': 2, 'All': 1, 'models': 1, 'wrong,': 1, 'but': 1, 'some': 1, 'useful': 1})
    ```

Stemming

- ***Stemming*** is the process of reducing inflected words to their word stem, base or root form, generally a written word form.



[Source: https://devopedia.org/stemming]

- Example:

```
from nltk.stem import PorterStemmer
words = ["walk","walks","walked","walking","walker"]
ps = PorterStemmer()

for w in words:
    print(ps.stem(w))
```

```
walk
walk
walk
walk
walker
```

Lemmatizing

- ***Lemmatizing*** is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.
- It considers the context and converts the word to its meaningful base form which is called Lemma.
- For instance, stemming the word "caring" would return "car".

```
nltk.download("wordnet")
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()

print("good :", wnl.lemmatize("good"))
print("better :", wnl.lemmatize("better", pos="a")) # adjective
print("best :", wnl.lemmatize("best", pos="a"))
```

```
good : good
better : good
best : best
```

5. Exercises

3.13 Playing with language – NLP lab

- Refers to “3.13 NLP.html”

3.21 Revision lab 1 – tokenize!

- Refers to “3.21 Nltk 01 - Tokenise.html”

3.22 Revision lab 2 – Finding root forms of words

- Refers to “3.22 Nltk 04 - Stemming.html”

3.23 Revision lab 3 – pre-processing text

- Refers to “3.23 Nltk 05 – Pre-processing Text.html”

3.201 Regular Expressions

- Refers to “3.201 Topic 3 – lab 2 regex.html”

6. Practice Quiz

- Work on *Practice Quiz 03* posted on Canvas.

Useful Resources

-
  - http://