MECG11503/MEME19803 Programming for Data Analytics Topic 1: Understanding Basic Programming

Dr Liew How Hui

Jan 2023

Outline

Background

Data science (or data analytics) is about drawing useful conclusions from large and diverse data sets through exploration, prediction, and inference.

- It became popular lately because everyone is trying to immitate popular online platforms (Facebook, Amazon, Neflix, Google) to *mine information from data* for target advertisement.
- It is actually old (using the term ‘data mining’) and has a long history related to the development of statistics.
- It is not a panacea to all problems.

Background (cont)

- Data Mining: Data + Rules $\Rightarrow$ Output
- Data Science: Data + Output $\Rightarrow$ Rules

	Data Mining	Data Science
Data Sources	Structured Data from RDBMS	Unstructured Data (log files, audio, images, emails, tweets, raw text, documents)
Tools	Data Visualisation, Statistics	Machine Learning
Goals	Knowledge discovery, Decision making	New business value with new functionality (e.g. MS Teams for Online learning, etc.)

Course Outcomes of this Subject

CO1: Demonstrate programming development to turn data into a format suitable for a data science pipeline.

(Topic 1. Group Assignment 20% → A3)

(Topic 4. Assignment 2 (5%) → A3)

CO2: Interpret data using exploratory data analysis. (Topic 2. Practical Quiz 10% → C5)

CO4: Create graphical representations of data.

(Topic 6. Assignment 2 (15%), Practical 2 (15%). By Dr Goh)

CO3: Manipulate data by creating new features, reducing dimensionality, and by handling outliers in the data.

(Topic 3 and Topic 5. Assignment 20%, Practical 3 (15%). By Dr Ivan Yeo)

Class Arrangements

- Week 1: No issue with Monday
- Week 2: Monday is CNY Second Day, change to 6pm–9pm Sunday 29th January 2023
- Week 3: No issue with Monday
- Week 4: Monday is a replacement holiday for Thaipusam, class (and practical quiz) continue as usual.

Outline

Programming Data Structures

Programming Data Structures (stored in memory):

- Basic/Simple Data Structures:
  - Boolean data types
  - Integral data types
  - Floating point data types
  - String data types
- Container Data Structures: Tuple, List, Set, Dictionary (Hashtable, Associative array)
- Record (or more general, Object-Oriented) Data Structures: dataclass, class, etc.

File Data Structures

Real-world data are **compound** (**complex**) **file** data structures (stored in harddisk, SSD or cloud files):

- Word (.doc, .docx), Excel (.xls, .xlsx), Powerpoint (.ppt, .pptx): containing file attributes, text paragraphs, images, tables, etc.
- PDF documents (.pdf)
- Images (.bmp, .jpg, .png, .webp)
- CSV, HTML, JSON, etc.

They are read and stored as record data structures which can be “broken down” into a collection of smaller record data structures (containing even smaller record data structures) with ‘basic data structures’ as the smallest components.

Basic Data Structures (cont)

Basic data type — Boolean (`bool` in Python)

- Values: True, False
- Logical connectives: and, or, not

Application:

- Predicates: They are functions that return Boolean values (and are introduced in discrete maths). E.g. quadraticdiscriminant(a,b,c) := $b^2 - 4ac > 0$
- Predictes or compound predicates are used in if statements, while loops, etc.

Basic Data Structures (cont)

Basic data type — Integer (`int` in Python)

- Values: ..., -3, -2, -1, 0, 1, 2, 3, ... (can be arbitralily large in Python but not Numpy array)
- Arithmetic Operations: +, -, *, /, //, **, %
- Relational Operations: >, >=, ==, ! =, <=, <
- Bit Operations: <<, >>, &, |,^

Applications

- Representing ordinal data
- Counting
- Indexing in a for loop (and general indexing)

Beware: Python integer is bignum is other system. Numpy has fixed-bit integers int16, int32, int64.

Basic Data Structures (cont)

Basic data type — Floating-point Number (`float` in Python)

- Scientific notation: $\pm$ X.DDDDD $\times 10^{y}$. E.g speed of light `c = 2.99792458e+08` ms^{-1}
- Arithmetic Operations: +, -, *, /, **
- Relational Operations: >, >=, ==, ! =, <=, <
- math library

Applications:

- Approximating continuous numbers in Physics, engineering, financial problems

Basic Data Structures (cont)

Basic data type — String (`str` in Python)

- `"a string"`, `'a string'`, `"""multiline strings"""`
- Operations: `+`, len(), `"repeat"*5`, `.lower()`, `.upper()`, `.capitalize()`, ...

Applications:

- Unicode text data represention
- “categorical data”, e.g. blood types (A, B, AB, O), etc.

Remark: Byte strings are lower level than strings in Python and will be skipped.

Tuple Container

Tuple: (a,b,c)

Application:

- Simple assignment: a,b,c = 1,2,3
- Encoding error: (result, error)
- Construct array of certain shape: np.zeros((2,2,3))

List Container

List : [a1, a2, a3, ...]. It is super-powerful but slow.

Application:

- Store arbitrary items: [”string”, 1, 3.14, True]
- Programming: [], append, len
- Construct Numpy array: E.g. np.array([[0.3,0.5],[-0.2,0.7]])
- Construct trees

Set Container

Set: {a1, a2, a3, ...}. It works like a finite set in maths.

Application:

- Find $A \cap B$, e.g. A=Customers buying A, B=Customers buying B
- Not really a 'set' in the mathematical sense because a set of empty set is not allowed in Python but allowed in maths.

Containers (cont)

Dictionary / Associative Map: {key1:val1, key2:val2, ...}

Application:

- Key-value pair: {key : value}, mydict[”word”] = 1
- Basic idea for key-based NoSQL database
- Count items, e.g. mydict[”word”] += 1
- Construct data frame

Record Data Structures

Example: customer information record:

```
from dataclasses import dataclass

@dataclass
class CustomerRecord:
    Name : str
    Age  : int      # or float? Is 21.8 years old OK?
    Address : str
    Phone : int     # or string?

#CustomerRecord(Name="Ali", Age=18, Address="KL",
                Phone=12345678)
print(CustomerRecord("Ali", 18, "KL", 12345678))
```

Recursive Record Data Structures

Example: Binary Tree

```
class BinTree:
    def __init__(self,node=None, left=None, right=None):
        self._node  = node
        self._left  = left
        self._right = right
    def __str__(self):
        if self._node is None:
            return ""
        elif self._left is not None and self._right is not None:
            return str(self._node) + \
                   "<ltree>" + str(self._left)  + "</ltree>" + \
                   "<rtree>" + str(self._right) + "</rtree>"
        elif self._left is not None:
            return str(self._node) + \
                   "<ltree>" + str(self._left) + "</ltree>"
        elif self._right is not None:
            return str(self._node) + \
                   "<rtree>" + str(self._right) + "</rtree>"
        else:
            return str(self._node)

print(BinTree())
print(BinTree("+",BinTree(1),BinTree(2)))
print(BinTree("+",BinTree("-",BinTree(3),BinTree(5)),BinTree(2)))
```

Even more complex examples: XML tree, graph, ...

Outline

Program Flow Structures

'Program Flow' is sort of like 'cooking process' for computer.

- Ingredients: data (of particular data structures)
- Operations: functions — can take data and can return data
- Steps: A sequence of steps (programming terminology: statements), a step can be an assignment statement, applying functions, if-else statements, for loops, while loops, etc.

Functions

In a programming language, a function may take in value(s) and may return a value.

Example 1: A function that only return a value: os.getcwd(); a function that does not return a value: os.chdir("d:/")

Example 2: Mathematical functions, e.g. sin, cos, exp, etc. from the `math` library. They takes a floating number and return a floating number

Example 3: `len` can be used find the length of some lists or sets. E.g. len(set(['a', 'A', 'a', 'b']).union([1,'a',3]))

Example 4: Predicates takes in values and returns a boolean. E.g. def is_positive(x): return $x > 0$

Functions (cont)

User-defined functions are just functions defined by a user following the Python programming syntax. For example, if we want to define a function which calculates sine using the degree as unit, then, we can define our own sine function using the 'sin' from math library to do the real computation:

```
def sine(Degree):
    from math import sin, pi
    return sin(Degree*pi/180.0)
```

Program Flows

The various data structures and functions can be orgnised in different programming paradigms to form a program flow structure / architecture:

- Imperative Programming
- Object-Oriented Programming (Complicated for beginners)
- Functional programming (Complicated even for programmers? Welcome by math enthusiast?)

Imperative Programming

It is based on the ‘change of states’ and the following structures:

- if statement : decision
- for loop : going through a list
- while loop : indefinite loop
- break and continue
- defining functions

Examples of Imperative programming languages: C, C++, Python, Go, Fortran, Pascal, MODULA2, MODULA3, ALGO 60, ADA, COBOL, etc.

Imperative Programming (cont)

Example: Simple linear regression:

$$y_i = ax_i + b, \quad i = 1, \ldots, n.$$

The estimate for a and b are as follows (given in SPM and university year 1 statistics):

$$a = (n \sum x_i y_i - \sum x_i \sum y_i)/(n \sum x_i^2 - (\sum x_i)^2)$$
$$b = (\sum x_i^2 \sum y_i - (\sum x_i y_i)(\sum x_i))/(n \sum x_i^2 - (\sum x_i)^2)$$

Imperative Programming (cont)

Implementation using Year 1 Programming Technique in Python:

```
1  import pandas as pd
2  # https://github.com/llSourcell/linear_regression_demo
3  df = pd.read_csv("brain_body.txt")
4  Y  = df.iloc[:,1].values
5  X  = df.iloc[:,0].values
6
7  if X.shape != Y.shape:
8      print(f"Error length(X)={len(X)} != length(Y)={len(Y)}")
9  n = Y.shape[0]; D = range(n)
0  SX = SY = SX2 = SXY = 0.0
1  for i in D: SX  += X[i]
2  for i in D: SY  += Y[i]
3  for i in D: SX2 += X[i]**2
4  for i in D: SXY += X[i]*Y[i]
5  denom = n * SX2 - SX*SX
6  a = (n * SXY - SX*SY)/denom
7  b = (SX2*SY - SXY*SX)/denom
8  print("a=",a,"\nb=",b)
```

Imperative Programming (cont)

Implementation using Year 1 Programming Technique in C++:

```
1  #include <iostream>
2  // https://www.mlpack.org/doc/mlpack-3.0.4/doxygen/formatdoc.html
3  #include <mlpack/core.hpp> // g++ -fopenmp default1a.cpp -lmlpack
4
5  int main() {
6      arma::mat df;
7      // https://github.com/llSourcell/linear_regression_demo
8      mlpack::data::Load("brain_body.txt", df);
9      //df.print();
0      std::cout << df.n_rows << " x " << df.n_cols << "\n";
1      arma::vec Y = arma::conv_to<arma::vec>::from(df.row(1)); // body
2      arma::vec X = arma::conv_to<arma::vec>::from(df.row(0)); // brain
3      //X.print();
4
5      const int i0 = 1; // first element in the data is string
6      int n = Y.n_elem-i0, i;
7      double SX = 0.0, SY = 0.0, SX2 = 0.0, SXY = 0.0;
8      for(i=i0; i<n+i0; i++) { SX  += X[i];      }
9      for(i=i0; i<n+i0; i++) { SY  += Y[i];      }
20     for(i=i0; i<n+i0; i++) { SX2 += X[i]*X[i];   }
21     for(i=i0; i<n+i0; i++) { SXY += X[i]*Y[i]; }
22     double denom = n * SX2 - SX*SX;
23     double a = (n * SXY - SX*SY)/denom;
24     double b = (SX2*SY - SXY*SX)/denom;
25     std::cout << "a=" << a << "\nb=" << b << "\n";
26 }
```

Object-Oriented Programming

It is based on classes and objects and it usually includes the imperative programming structures (otherwise many programmers find it difficult to understand a pure object-oriented programming language):

- Classes = Data + Methods (Functions)
- Objects: Instances of Classes
- Instantiate an object: obj = NewClass()
- Sending 'message' to an object using a method: obj.method(parameters)

Examples of object-oriented programming languages: C++, Java, Kotlin, Python, Ruby, Smalltalk, Objective-C, Swift, etc.

Object-Oriented Programming (cont)

Implementation using Year 2 Object-Oriented Programming in Python:

```
1  import pandas as pd
2  # https://github.com/llSourcell/linear_regression_demo
3  df = pd.read_csv("brain_body.txt")
4  Y  = df.iloc[:,1].values
5  X  = df.iloc[:,0].values
6
7  n = Y.shape[0]
8  denom = n*(X**2).sum()-X.sum()**2
9  a = (n * (X*Y).sum() - X.sum()*Y.sum())/denom
0  b = ((X**2).sum()*Y.sum() - (X*Y).sum()*X.sum())/denom
1  print("a=",a,"\n","b=",b)
```

Note: `df` belongs to the 'pandas.core.frame.DataFrame' class.

Object-Oriented Programming (cont)

High-level object-oriented programming allows simple Data Processing in Python because many things are **wrapped** inside objects:

```
1  import pandas as pd, numpy as np, statsmodels.api as sm
2  # https://github.com/llSourcell/linear_regression_demo
3  df = pd.read_csv("brain_body.txt")
4  Y  = df.iloc[:,1].values
5  X  = df.iloc[:,0].values
6
7  X = sm.add_constant(X)
8  model = sm.OLS(Y, X)  # Create Ordinary Least Square object
9  linreg = model.fit()  # Use the method .fit() to fit data
0  print(linreg.params)  # print the 'public' values
1  print(linreg.summary())  # Call the summary() method
2
3  import matplotlib.pylab as plt
4  b, a = linreg.params  # Linear Regression: Y = aX + b
5  plt.plot(X, Y, '+')
6  Xrange = np.linspace(X.min(),X.max(), 100)
7  plt.plot(Xrange, a*Xrange+b,'-'); plt.plt.show()
```

Functional Programming

It is based on functions and it usually difficult because it usually does not include the imperative programming structures:

- Data are similar to imperative programming language (basic and record data structures) but are immutable in general.
- List is a fundamental data structure
- Functions works on data structures recursively most of the time

It is so closely related to discrete mathematics that Examples of functional programming languages: Haskell, Ocaml, SML, Scheme, Racket, etc.

Outline

Data Processing Components

- Data Frame (will be introduced in Topic 3 again with advanced features by Dr Ivan Yeo): 'Collections' of Columns (1-D arrays)
- Basic Statistical Models: Simple numeric statistics in Topic 2. More basic statistics in Topic 3.
- Complex Statistical Models: Predictive Modelling
- Dashboards
- Report Generation
- Powerpoint Presentation

Python Software Environments

- Python Shell + Text Editor
- Spyder IDE
- Jupyter Notebook: Popular but I don't recommend

Practical 1: Reading data of various formats (using software libraries).

1. To familiarise with the program development environment to edit, save and execute programs.
2. To read data of various formats using Python's libraries.

Software Libraries

Python's standard libraries:

- csv, sqlite3, lxml, json, ...: Can handle specific structured and semi-structured data.
- struct (for interpreting bytes as packed binary data), codecs (codec registry and base classes): for general (totally unstructured) data.
- re: Can be helpful in text processing

Python's Extra / Addon libraries:

- pandas (Topic 3, Dr Ivan Yeo): for loading structured (tabular) data
- PIL (pillow image library): for loading images of specific types

Unstructured Data

Examples:

- text (variable length)
- images
- documents: HTML, XML, SGML, Word, PDF
- audios
- videos
- game data

Structured Data

Examples:

- Company CSVs, Excels (.xls, .xlsx, .xlsb, .xlsm, .xltx, .xltm)
- SQL database

Computer Representations:

- Excel Table ???
- SQL Database: Data definition (schema)
- Numpy Array (Topic 2)
- Data Frame: R's data.frame (1991), Python's pd.DataFrame (11 Jan 2008)

Structured Data (cont)

Python libraries to work with structured data:

```
1 import numpy as np
2 import openpyxl # read Excel 2010 files
3 import xlsxwriter # write Excel 2010 files
4 import xlrd, xlwt # read/write Excel .xls files
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 import statsmodels.api as sm
```

Structured Data (cont)

Reading CSV using Python's csv library:

```
1  ### https://docs.python.org/3/library/csv.html
2  import csv, numpy as np
3  alist = []
4  # https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
5  with open('Concrete_Data.csv', newline='\n') as f:
6      csvreader = csv.reader(f, delimiter=',', quotechar='"')
7      for i, row in enumerate(csvreader):
8          # First row is header
9          if i==0:
0              header = []
1              for item in row:
2                  header.append(item)
3              continue
4          floatlist = []
5          for item in row:
6              floatlist.append(float(item))
7          alist.append(floatlist)
8      tbl = np.array(alist)
9  print(header)
0  print(tbl)
```

Structured Data (cont)

Python's pandas library (details by Dr Ivan Yeo in Topic 3) read the CSV file much cleaner compare to the use of `csv` library:

```
1 import pandas as pd
2 fn="Concrete_Data.csv"
3 pop_data = pd.read_csv(fn)
4 print(pop_data.head(4))
5 print(pop_data.tail(4))
6 print(pop_data.index)
7 print(pop_data.columns)
8 print(pop_data.dtypes)
```

Structured Data (cont)

HTML with Tables:

```
<html>
<body>
Table 1
<table>
  <tr><th>Firstname</th><th>Lastname</th><th>Age</th><th>Salary</th></tr>
  <tr><td>Jill</td><td>Smith</td><td>35</td><td>3,000--6,000</td></tr>
  <tr><td>Eve</td><td>Jackson</td><td>?</td><td>6,000--9,000</td></tr>
</table>

Table 2
<table>
  <tr><th>Name</th><th colspan="2">Telephone</th></tr>
  <tr><td>Bill Gates</td><td>55577854</td><td>55577855</td></tr>
</table>
</body>
</html>
```

Structured Data (cont)

Python libraries to read HTML Tables:

```
import pandas as pd
tbls = pd.read_html("tableeg.html")   # requires lxml
for i,tbl in enumerate(tbls):
    print((f" Table {i+1} ").center(70, "="))
    print(tbl)
    print()
```

Structured Data (cont)

JSON = JavaScript Object Notation

Sample 1 (by columns?):

```
{
    "Firstname" :    ["Jill", "Eve"],
    "Lastname" :     ["Smith", "Jackson"],
    "Age" :          [35, "?"],
    "Salary" :       ["3,000--6,000",
                      "6,000--9,000"]
}
```

Structured Data (cont)

Sample 2 (by rows?):

```
[
{
    "Firstname" :    "Jill",
    "Lastname" :     "Smith, II",
    "Age" :          35,
    "Salary" :       "3,000--6,000"
},
{
    "Firstname" :    "Eve",
    "Lastname" :     "Jackson",
    "Age" :          "?",
    "Salary" :       "6,000--9,000"
}
]
```

Structured Data (cont)

Pandas library supports JSON data:

```
import pandas as pd
df1 = pd.read_json("sample1.json")
print(df1)
df2 = pd.read_json("sample2.json")
print(df2)
```

Structured Data (cont)

SQLite database: Use sqlite & SQL language
or
Python sqlite3 library to read sqlite database

```
from pandas.io import sql
import sqlite3
conn = sqlite3.connect('data.db')
query = 'SELECT * FROM tablename'
tbl = sql.read_sql(query, con=conn,
                   parse_dates={'date': '%d/%m/%Y'})
print(tbl.head())
```

Structured Data (cont)

SQLite Database only supports 5 basic data types:

- NULL. The value is a NULL value.
- BLOB. The value is a blob of data, stored exactly as it was input.
- TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- INTEGER. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.

Structured Data (cont)

Other formats:

- SPSS (requires special API from IBM),
- SAS (?)
- Advanced / Commercial SQL Database (requires authentication)
- NoSQL Database (Datalog Query? SPARQL?)
- SAP (?)

Structured Data (cont)

Advanced / Commercial SQL Database supports more data types. E.g. Postgresql (see `http://www.postgresqltutorial.com/postgresql-data-types/`) has richer data structures:

- Boolean;
- Character types such as char, varchar, and text;
- Numeric types such as integer and floating-point number;
- Temporal types such as date, time, timestamp, and interval;
- UUID for storing Universally Unique Identifiers;
- Array for storing array strings, numbers, etc.;
- JSON stores JSON data;
- hstore stores key-value pair;
- Special types such as network address and geometric data.

Structured Data (cont)

Numpy array and Panda dataframes:

- dtype / pd.DataFrame().dtypes
  - b boolean
  - i signed integer
  - u unsigned integer
  - f floating-point
  - c complex floating-point
  - m timedelta
  - M datetime
  - O object
  - S (byte-)string
  - U Unicode
  - V void
- Array integer (32 bits, 64 bits) is not the same as Python integer
- Floating point numbers 32 bits and 64 bits

Structured Data (cont)

```
1  df = pd.DataFrame({
2     'a' : [1,2,3],
3     'b' : [4,5,6],
4     'c' : [7,8,9],
5  })   # default index = 0,1,2,...
6
7  df = df.rename(columns={'a': 'A', 'b' : 'B'})
8  df.dtypes
9  # Changing the data type
0  df['B'] = df['B'].astype("str")
```

Outline

Group Assignment (20%)

Introduction to the Group Assignment:

- Two-member/three-member group assignment.
- Reading original data using Python and
- No preprocessing of data using Excel or any other tools allowed!

Outcome of the Assignment: Demonstrate programming development to turn data into a format suitable for a data science pipeline.

Submit two reports — one **group report, 17%** (PDF), one **individual report, 3%** (hand-written).

Group Assignment (20%) cont.

The group report need to contain

- Background analysis of features and targets in each data
- Using Python to read the data array / table
- Using Python to summarise the statistics of the data
- Explain what sort of applications would be associated with the data and the sort of relevant data pipeline(s).

The individual report need to contain

- Summary and comment on the group report.
- Suggestions for the data you are interested in and how can the analysis of the suggested data will be helpful in your career.