NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Dictionaries; Text Files

Pedro Barahona
DI/FCT/UNL

Computational Methods

1st Semester 2021/22

Dictionaries

- Arrays (vectors, matrices, or higher dimension) are very convenient structures to organize numerical information, since each “cell” should contain a number.
- In many cases, information is not only numeric, e.g. it includes text (we do not consider other types of information, such as visual or sound or video).
- In Python, such information can be grouped in lists, that may be composed of elements with different data types. However, the access to the elements of a list requires numerical indices, and this is somehow unnatural and inconvenient.
- Take for example the information about the employees of a certain company. For each employee we may consider:
  - **id** – integer, representing a unique identification number in the company
  - **name** – text, with the name of the employee
  - **date** – text, in format YYYY-MM-DD, representing the date of employment
  - **salary** – real number, representing the monthly salary of the employee

id	**name**	**date**	**salary (€)**
98	Rui Silva	2011-10-23	1654.3

Dictionaries

- Although complex information is better maintained in a database, in simple applications, this heterogeneous information may be organized in a **dictionary**.
- Dictionaries are a special case of sets, in that they have elements that are not sorted by any index. Nevertheless, elements may be indexed, similarly to vectors, with some main differences.
  - Elements of a dictionary are **key:value** pairs;
  - Unlike vectors (but like lists), values may be of different data types;
  - **Items** are indexed by **keys** that are strings (not numeric indices);
  - **Values** are accessed by specifying their **keys**.

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Example:

- An employee, **emp**, may be represented by a dictionary, which is a set (curly brackets) whose elements have the form **key:value**)

```
emp = {'id':98, 'name':'Rui Silva', 'date':'2011-10-23', 'salary':1654.3}
```

Dictionaries

- Several methods are defined for instances of the class dictionary that allow their manipulation.
- As sets, dictionaries may be created empty or by explicit specification of their elements.
- At any time, dictionaries may be cleared of all their elements

```
In : emp = {}
In : emp
Out: {}
In : emp = {'id': 44, 'name': 'Rui Silva'}
In : emp
Out: {'id': 44, 'name': 'Rui Silva’}
In : emp.clear()
In : emp
Out: {}
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Dictionaries

- Once created the elements (items) of the dictionary may be accessed or changed by methods **get()** and **__setitem__()**.
- They can be called with the more usual [ ] notation.

```
In : emp = {'id': 67, 'name': 'Silva'}
In : emp
Out: {'id': 67, 'name': 'Silva'}
In : emp.get('id')
Out: 67
In : emp['name'] = "Rui Silva"
In : emp. __setitem__('id’, 98)
In : emp['name']
Out: 'Rui Silva'
In : emp
Out: {'id': 98, 'name': 'Rui Silva'}
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Dictionaries

- Items may be added to or deleted (and retrieved) from dictionaries with methods **update()** and **pop()**, respectively.
- Method **popitem()** deletes and retrieves, nondeterministically, some item of the dictionary

```
In : emp = {'id’: 98', 'name': 'Rui Silva’}
In : emp.update({'date':'2011-10-23', 'salary': 1654.3})
In : emp
Out: {'id':98,'name':'Rui Silva','date':2011-10-23','salary':1654.3}
In : emp.pop('salary')
Out: ('salary', 1654.3)
In : emp
Out: {'id':98,'name':'Rui Silva','date':2011-10-23'}
In : emp.pop('id')
Out: ('id', 98)
In : emp
Out: {'name':'Rui Silva','date':2011-10-23'}
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Dictionaries

- The values and the keys of a dictionary can be obtained by methods **keys()**, **values()**, and **items()**.
- The information is returned in lists that can be subsequently iterated

```
In : emp
Out: {'id':98,'name':'Rui Silva','date':'2011-10-23'}
In : emp.keys()
Out: dict_keys(['id', 'name', 'date'])
In : emp.values()
Out: dict_values([98,'Rui Silva','2011-10-23']
In : emp.items()
Out: dict_items([('id':98),('name':'Rui Silva'),('date':'2011-10-23')]
In : for k in emp.keys():
...:     print(emp.get(k))
Out:
98
Rui Silva
2011-10-23
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Dictionaries

- Finally, dictionaries may be copied,
    - either in full; or only
    - its structure is created with some of their keys and None values.

```
In : emp
Out: {'id':98,'name':'Rui Silva','date':'2011-10-23'}
In : x = emp.copy()
In : x
In : {'id':98,'name':'Rui Silva','date':'2011-10-23’}
In : ks = ('id', 'date')
In : y = emp.fromkeys(ks)
In : y
Out: {'id': None, 'date': None}
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3

Tables

- Many applications require to maintain several records in a “table”, i.e. a set of records of the same type, possibly indexed by a number or a “key”.

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- Complex applications, requiring several tables, should of course be supported in databases. Nevertheless, for simple applications, tables can be directly modelled by more general programming languages.
- As discussed, heterogeneous records can be modelled in Python with dictionaries.
- Dictionaries may be grouped together in a table, typically as
    - lists (if they are to be indexed by numbers); or
    - dictionaries (if indexed by more general keys).

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Tables: Lists of Dictionaries

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- When tables are organised as lists, the usual list operations and methods can be used to handle such tables, taking into account that elements are dictionaries.

```
In : emps = []
In : emp ={'id':98,'name':'Rui Silva','date':'2011-10-23','salary':1654.3}
In : emps.append(emp)
In : emp ={'id’:56,'name':’Maria Santos','date':'2008-12-18','salary':1742.4}
In : emps.append(emp)
In : emp ={'id’:43,'name':Carlos dias','date':'2003-04-12','salary':2017.6}
In : emps.append(emp)
In : emp ={'id’:12,'name':Isabel Rio','date’:'1987-09-05','salary':2916.8}
In : emps.append(emp)
In : emps[2]['salary’].  # 2 is the element position in the list
Out: 2017.6
```

Tables: Dictionaries of Dictionaries

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

- When tables are organised as dictionaries of dictionaries, a key of the latter can be used as the key to the whole table (possibly duplicated in the records).

```
In : emps = {}
In : emp ={'id':98,'name':'Rui Silva','date':'2011-10-23','salary':1654.3}
In : emps.update({98:emp})
In : emp ={'id’:56,'name':’Maria Santos','date':'2008-12-18','salary':1742.4}
In : emps.update({56:emp})
In : emp ={'id’:43,'name':Carlos Dias','date':'2003-04-12','salary':2017.6}
In : emps.update({43:emp})
In : emp ={'id’:12,'name':Isabel Rio','date’:'1987-09-05','salary':2916.8}
In : emps.update({12:emp})
In : emps[98]['salary’]   # 98 is the id of the dictionary to be accessed
Out: 1654.3
```

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Lists of Dictionaries

- The following function projects a table (a dictionary of dictionaries) into some its columns, though iteration on their elements.

```
def emps_short(emps,keys):
    """creates a dictionary of dictionaries, by
    projecting the dictionaries into the given keys"""
     shorts = {}
     for key_1 in emps:
         emp = emps[key_1]
         short = {}
         for key_2 in keys:
             short.update({key_2:emp[key_2]})
         shorts.update({key_1:short})
     return shorts
```

id	name	date	salary (€)
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
In : ks = ('id', 'date’)
In : shorts = emps_short(emps,ks)
In : shorts[43]
Out: {'id':43,'name':Carlos Dias’}
```

File Input / Output

- When the amount of data is large, it is not practical/feasible to enter data and read program results from the terminal. In most cases, we use files to have permanent access to this data (here we will only consider text files – that can be read by any text processor, such as notepad).
- Files are managed by a file system (part of the operation system – Windows, Linux, MacOS) and files are organised in a (inverted) tree.
- At the top there is a root directory that recursively contains other directories (the branches of the tree) and possibly files (the leafs of the tree).
- Spyder supports some typical file system instructions, that can be used either in a program or at the terminal. Among the most useful
  - **pwd** – returns a string representing the current directory
  - **ls** – shows the files and folders in the current directory
  - **cd name** – changes the current directory to the directory with name
  - **cd ..** – changes the current directory to its parent directory
  - **cd //** – makes the root as the current directory

File Input / Output

- To read to or write from a file, it is necessary a) to **open** it, and after handling its data (reading from / writing into), the file should be **closed**.
- In **Python**, opening a file is done with instruction
  - **`open(fileName, mode)`**

  where
  - **fileName** is the name of the file (as seen from the current directory)
  - **mode** is either “r” for read or “w” for write

```
fid = open('file.txt', 'r')
```

- The function returns an object (the file handler) that should be subsequently used to read/write data and finally to close the file.

File Input / Output

- The function returns an object (the file handler) that should be subsequently used to read/write data and finally to close the file.
  - **Note:** If the file could not be opened, the function returns an error. To avoid aborting the computation this error should be handled by an IO exception

```
try:
    fid = open('file.txt', 'r')
except IOError:
    print(Error: no such file’)
```

- Once used, the file should be closed with method
  - **`fid.close()`**

  where
  - **fid** is the file handler that was obtained when the file was opened.

File Output

- The access to an open file is **sequential**, i.e. data items are read/written one after the other with no going back or direct access to some kth item of the file.
- To write (text) data in a file, previously opened the method write should be used on the fid object.

```
In : fid = open('example.txt', 'w')
In : fid.write('This is the first line;\nand this is the second.\n')
Out: 48
In : fid.write('Fim\n')
Out: 4
In : fid.close()
```

example.txt

```
This is the first line;
and this is the second.
Fim.
```

- Note the explicit use of the new line (\n) character.
  - there is no writeln method in Python

File Input

read()

- To read a file, the method read may be used.
- This method reads the whole file (from the current position to the end) and retuns a string with all characters that were read, including the new lines.
- Reading beyond the end of file returns an empty string.

readlines()

- Quite often it is more useful to read the text file line by line, so as to process the information in each line
- The method readlines() returns a list with all the file lines.

readline()

- To read incrementally the file, the method readline() reads a single line (from the current position of the cursor).
  - It returns an empty string if attempting to read **beyond** the end of the file.

File Input

read()

- To read a file, the method read may be used.
- This method reads the whole file (from the current position to the end) and retuns a string with all characters that were read, including the new lines.
- Reading beyond the end of file returns an empty string.

readlines()

- Quite often it is more useful to read the text file line by line, so as to process the information in each line
- The method readlines() returns a list with all the file lines.

readline()

- To read incrementally the file, the method readline() reads a single line (from the current position of the cursor).
  - It returns an empty string if attempting to read **beyond** the end of the file.

File Input / Output

- Example: Read the file with a matrix and return (it as a lists of lists)

```python
def read_matrix(fname):
    """returns a matrix stored in file"""
    fid = open(fname, 'r');
    mat = []
    lines = fid.readlines();
    fid.close()
    for line in lines:
        row = []
        numbers = line.strip().split(' ');
        for number in numbers:
            row.append(int(number))
        mat.append(row)
    return mat
```

matrix.txt

```
12 20 30 89
34 50 98 13
25 47 26 56
```

```
In : mm = read_matrix('matrix.txt')
In : mm
Out: [[12, 20, 30, 89], [34, 50, 98, 13], [25, 47, 26, 56]]
```

Handling Dictionaries from Files

- Typically, the information contained in a table is stored in a file, given the large volume of data it contains.
- Of course, processing this data directly from the file is very inefficient since
  - Access to files is sequential.
    - Once read say element i, reading element i-1 requires reading the file again.
  - Access to files is slow:
    - Although disks nowadays are much faster than some years ago, namely the SSD disks that are fully electronic and have no mechanical components, its access is typically at least one order of magnitude slower than that to RAM memory, that have better channels to the CPU.
- Hence, processing data in a table, is done in up to 3 steps:
  1. Reading the table from a file to list or dictionary of records (dictionaries)
  2. Process the data, e.g. changing elements of the table
  3. Write the new table into a file

Reading Dictionaries from Files

- We illustrate the reading of a table from a file with a table created in CSV format (possibly from a spreadsheet as EXCEL) where
  - the first line indicates a title
  - the second line indicates the name of the fields (record keys);
  - the other lines contain the information (values) of the records.

Table of Employees

id	name	date	salary
98	Rui Silva	2011-10-23	1654.3
56	Maria Santos	2008-12-18	1742.4
43	Carlos Dias	2003-04-12	2017.6
12	Isabel Rio	1987-09-05	2916.8

```
Table of Employees
id,name,date,salary
98,Rui Silva,2011-10-23,1654.3
56,Maria Santos,2008-12-18,1742.4
43,Carlos Dias,2003-04-12,2017.6
12,Isabel Rio,1987-09-05,2916.8
```

- In this case (a CSV file), all fields are separated by commas (‘,’)

Reading Tables of Dictionaries from Files

```
Table of Employees
id,name,date,salary
98,Rui Silva,2011-10-23,1654.3
56,Maria Santos,2008-12-18,1742.4
43,Carlos Dias,2003-04-12,2017.6
12,Isabel Rio,1987-09-05,2916.8
```

- Reading a file requires the following steps:
  1. Open the file in read mode
  2. Skip the first line
  3. Read the second line, striping it from leading and trailing spaces - strip() method;
  4. Obtain the keys of the dictionaries by splitting the line: split(‘,’) method;
  5. Read the following lines and close the file.
  6. Start an empty table;
  7. For each line read:
     1. Start an empty dictionary;
     2. Strip the line from leading and trailing spaces and split it in the commas;
     3. For each key in the keys:
        1. Add the corresponding value from the line to the dictionary
        2. “Advance” the line
  8. Return the table
- The code is shown in the next slide

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Reading Tables of Dictionaries from Files

```
def read_emps(fname):
    """reads a table, as a list of dictionaries from a file
    with name fname. The keys are named in the first line, and
    the data items in the subsequent lines, all separated by commas"""
    fid = open(fname, 'r')
    fid.readline()
    keys_line = fid.readline().strip()
    keys = keys_line.split(',')
    lines = fid.readlines()
    fid.close()
    emps = []
    for line in lines:
        line = line.strip().split(',')
        emp = {}
        for key in keys:
            emp.update({key:line[0]})
            line = line[1:len(line)]
        emps.append(emp)
    return emps
```

```
Table of Employees
id,name,date,salary
98,Rui Silva,2011-10-23,1654.3
56,Maria Santos,2008-12-18,1742.4
43,Carlos Dias,2003-04-12,2017.6
12,Isabel Rio,1987-09-05,2916.8
```

Writing Tables of Dictionaries into Files

- Writing a table to a file is done similarly.
  1. Open the file in write mode
  2. Write the title line
  3. For each key:
     1. Write the key and a trailing comma
  4. Replace the last comma by a new line
  5. Write the line
  6. For each emp in emps:
     1. Start an empty line;
     2. For each key in the keys:
        1. Add the corresponding value and a comma to the line
     3. Replace the last comma by a new line
     4. Write the line
  7. Close the file
- The code is shown in the next slide

```
Table of Employees
id,name,date,salary
98,Rui Silva,2011-10-23,1654.3
56,Maria Santos,2008-12-18,1742.4
43,Carlos Dias,2003-04-12,2017.6
12,Isabel Rio,1987-09-05,2916.8
```

Writing Tables of Dictionaries into Files

```
def write_emps(fname, emps):
    """reads a table, as a list of dictionaries from a file
    with name fname. The keys are named in the first line, and
    the data items in the subsequent lines, all separated
    by commas"""
    fid = open(fname, 'w')
    fid.write('Table of Employees\n')
    line = ""
    for key in emps[0].keys():
        line = line+key+","
    line = line[0:len(line)-1]+"\n"
    fid.write(line)
    for emp in emps:
        line = ""
        for key in emp.keys():
            line = line + emp[key] + ','
        line = line[0:len(line)-1]+"\n"
        fid.write(line)
    fid.close()
```

```
Table of Employees
id,name,date,salary
98,Rui Silva,2011-10-23,1654.3
56,Maria Santos,2008-12-18,1742.4
43,Carlos Dias,2003-04-12,2017.6
12,Isabel Rio,1987-09-05,2916.8
```

Tables with Dictionaries – Processing

- Once the table is read into memory, we can process it, namely finding information contained in it.

Some examples:

1. Find the average of the salaries of the employees;
2. Find the oldest employee (according to the dates);
3. Find the name of an employee with a given id
4. Restructure a table from a list of records to a dictionary of records
5. Write a file with the employees earning more than a certain amount;

  - In all cases, except the third, the table must be completely swept. In the third case, the sweeping should stop once the employee is found.

- These examples are left as exercises.