

LABORATORY OF DATA SCIENCE

File Data Access in Python

File Data Access in Python

2

- Basic functions and methods to manipulate files by default: **File object**
- **Open Method**
`open(file_name [, access_mode] [, buffering])`
- **Close Method**
 - ⌘ Flushes any unwritten information and closes the file object, after which no more writing can be done
 - ⌘ Python automatically closes a file when the reference object of a file is reassigned to another file. **Good practice to use the close() method to close a file!**
 - ⌘ **Syntax: fileObject.close()**

Example:

```
file=open("test.txt","w")  
print("Name: ", file.name)  
file.close()
```

Example:

```
with open("test.txt","w") as file  
    print("Name: ", file.name)  
file.close()
```

Open Method

3

```
open(file_name [, access_mode][, buffering])
```

- **file_name**: string value that contains the name of the file to be accessed
- **access_mode**: determines the mode in which the file has to be opened (read, write append etc.). This is optional parameter and the default file access mode is read (r)
- **buffering**: If the buffering value is set to 0, **no buffering** will take place. If the buffering value is 1, **line buffering** will be performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action will be performed with the indicated **buffer size**.

Open Method: Access Mode

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Open Method: Access Mode

wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

File Object Attributes

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.

Reading Files

- **Read Method** reads some quantity of data and returns it as a **string** (in text mode) or **bytes object** (in binary mode)

Syntax:

```
fileObject.read([size])
```

size: number of bytes to be read from the opened file. If *size* is missing or negative the entire file is read

Example:

```
file = open("test.txt", "r")
s = file.read()
print("The file contains: ", s)
file.close()
```

Reading Files

- Loop over the file object to read lines:

```
for line in file:  
    print(line)
```

- Read all lines of a file in a list

```
fileObject.readlines()
```

Example:

```
file = open("test.txt", "r")  
lines = file.readlines();  
file.close()
```

Writing Files

- **Write Method** writes the contents of string to the file, returning the number of characters written.
 - ⌘ The write() method does not add a newline character ('\n') to the end of the string

Syntax:

```
fileObject.write(string)
```

Example:

```
file = open("test.txt", "w")
file.write("Python is a great language.\r\nYeah its
great!!\r\n");
file.close()
```

Exercises on files

□ Exercise 1: RowCount.py

- ✘ Read the CSV file census_selected_header.csv
- ✘ Return:
 - The number of rows in the file
 - The minimum number of columns
 - The maximum number of columns

□ Exercise 2: Select.py

- ✘ Read the CSV file census_selected_header.csv
- ✘ Write in a CSV file census_cols.csv only the list of columns passed as input
- ✘ EX. of parameters:
 - census_selected_header.csv census_cols.csv 3 6 8

Columns to
be reported
in the output
file

Getopt - Command Line Option Parsing

- This function parses the argument **sys.argv** and returns a sequence of tuples containing **(option, argument)** pairs and a sequence of **non-option arguments**.

```
import getopt
```

```
opts, args=getopt.getopt(sys.argv[1:], "i:o:")
```

```
for opt in opts:
```

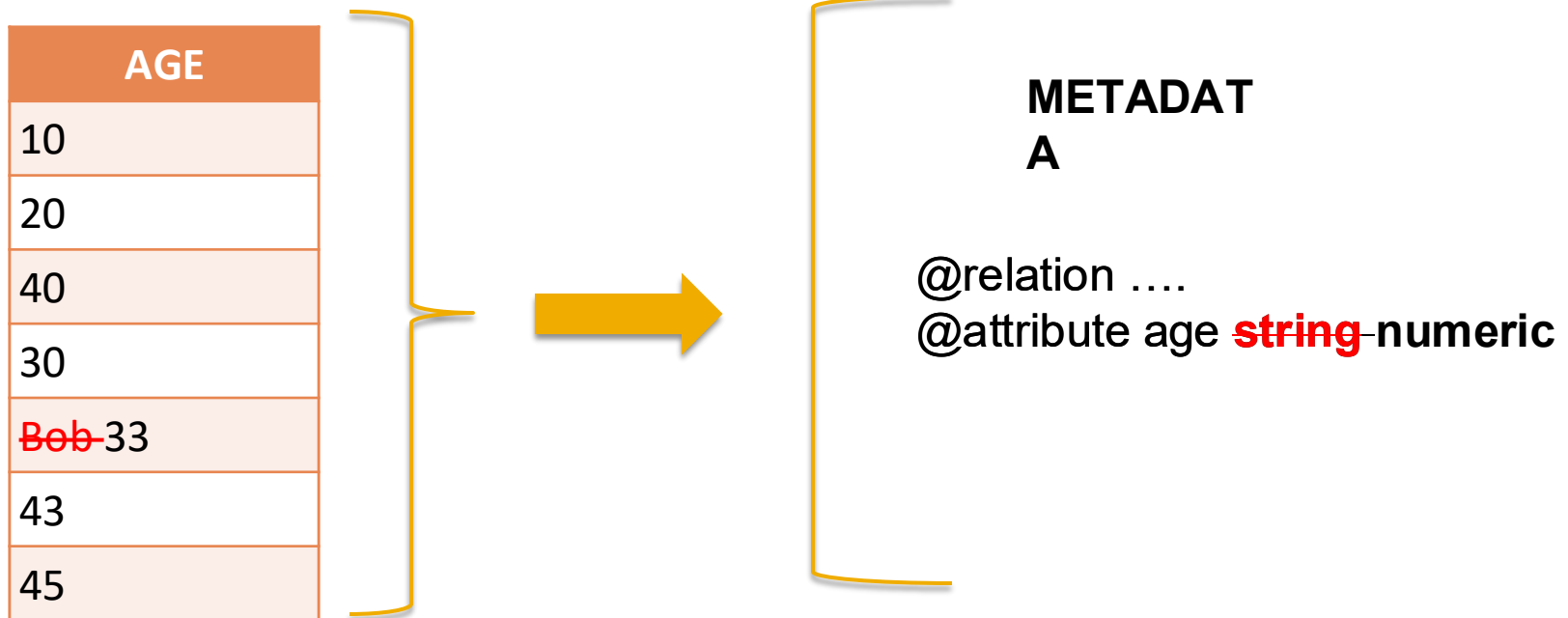
```
    print(opt)
```

1. sequence of arguments to be parsed, usually coming from **sys.argv[1:]** (ignoring the program name in `sys.argv[0]`)
2. option definition string for single character options. If one of the options requires an argument, **its letter is followed by a colon**.

Exercise: format conversions

12

- Write a Python program for converting a CSV file into the ARFF format

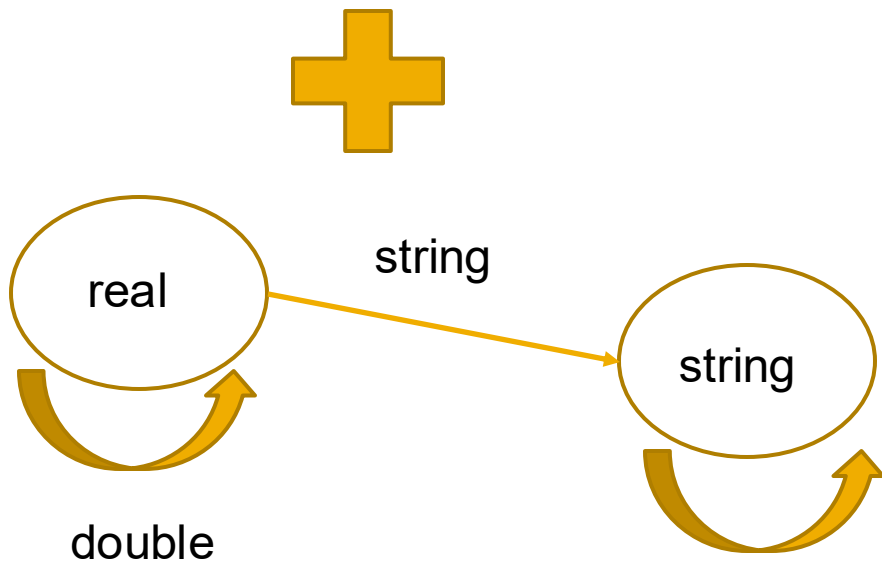


CSV2ARFF - solution

14

{ value1, value2, ..., valueK }
Set of values seen so far

enumerated $\leftarrow K \leq \text{Max?}$



real or string $\leftarrow K > \text{Max?}$

Double or string

XML File Data Access in Python

15

- Several interfaces for working with the Extensible Markup Language (XML)
- They are grouped in the **xml** package:
 - ✘ <https://docs.python.org/2/library/xml.html>
- The XML handling submodules are:
 - ✘ **xml.etree.ElementTree**: the ElementTree API, a simple and lightweight XML processor - <https://docs.python.org/2/library/xml.etree.elementtree.html#module-xml.etree.ElementTree>
 - ✘ **xml.dom**: the DOM API definition
 - ✘ **xml.dom.minidom**: a minimal DOM implementation
 - ✘ **xml.dom.pulldom**: support for building partial DOM trees
 - ✘ **xml.sax**: SAX2 (Simple API XML) base classes and convenience functions
 - ✘ **xml.parsers.expat**: the Expat parser binding

Exercise(s): format conversions

16

- Write a Python program for converting a CSV file into the ARFF format
- Write a Python program for converting a XML-elements file into a CSV file using a DOM parser
- Write a Python program for converting a CSV file into the JSON format
 - ✦ Assume that CSV files have a first row of meta-data with the names of the columns

JSON File Data Access

17

Object (Map)

Name (String)

Value
(String,Number)

Array

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Exercise(s): format conversions (3)

18

- Write a Python program for converting a CSV file into the ARFF format
- Write a Python program for converting a XML-elements file into a CSV file using a DOM parser
- Write a Python program for converting a CSV file into the JSON format
 - ✧ Assume that CSV files have a first row of meta-data with the names of the columns