PYTHON FOR DATA SCIENCE WORKING WITH DATA

Ana-María Dobre
based on EDX Course
SEP 2023

1. READ FILES WITH OPEN

Open file Example1.txt (1)
Notes:
- File1 is the file object
- "/files/examples/Example1.txt" = file path, including file name
- second parameter = mode ("r" for reading, "w" for writing and "a" for appending)

Use the file object to get info about the file: (2)

Always close the file object using method close: (3)

Using a with statement to open the file is better practice because it automatically closes the file: (4)

Note: file_stuff (the raw string) contains \n (so that Python knows to start a new line)

Output every line as an element in a list using the method readlines: (5)
Note: The method readline (singular) is used for reading the first line of the file.

We can use method readline twice! The first time it will read the first line of the file, the second time - the second line, etc.

We can use a LOOP to print each line individually as follows: (6)

We can specify the number of characters we would like to read from the string as an argument to the method readlines: (7)

```
(1)
File1 = open("/files/examples/Example1.txt", "r")

(2)
File1.name
File1.mode

(3)
File1.close()

(4)
with open("Example1.txt", "r") as File1:
    file_stuff=File1.read()
    print(file_stuff)
print(File.closed) # just a check
print(file_stuff) # you cannot read from File1 outside the indent. BUT YOU CAN PRINT THE FILE CONTENT

(5)
with open("Example1.txt", "r") as File1:
    file_stuff=File1.readlines()
    print(file_stuff)

(6)
with open("Example1.txt", "r") as File1:
    for line in File1:
    print(line)

(7)
with open("Example1.txt", "r") as File1:
    file_stuff=File1.readlines(11)
    print(file_stuff)
    file_stuff=File1.readlines(3)
```

2. WRITE FILES WITH OPEN

Create file Example2.txt, as follows (using open) (8)

Using the with statement, the code will run everything in the indent block, then closes the file: (9)

Same example as previous, but with a loop: (10)
#at the end of the loop, the file will be closed.

Append mode demo: (11)

Copy one file to a new file, like so: (12)

It is often very useful to know where the 'cursor' is in a file and be able to control it. The following methods allow us to do precisely this:
.tell() - returns the current position in bytes
.seek(offset,from) - changes the position by 'offset' bytes with respect to 'from'. From can take the value of 0,1,2 corresponding to beginning, relative to current position and end.

```
(8)
File2 = open("/files/examples/Example2.txt", "w")

(9)
with open("/files/examples/Example2.txt", "w") as File2:
    File2.write("Love is all around line 1\n")
    File2.write("And so the feeling grows line 2\n")

(10)
Lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
with open("/files/examples/Example2.txt", "w") as File2:
    for line in Lines
        File2.write(line)

(11)
with open("/files/examples/Example2.txt", "a") as File2:
    File2.write("Append this line\n")

(12)
with open("Example1.txt", "r") as readfile:
    with open("Example1.txt", "w") as writefile:
        for line in readfile:
            writefile.write(line)
```

3. PANDAS (I)

WORKING WITH & SAVING DATA

Pandas has the method unique to determine the unique elements in a column of a data frame: (21)

Create a new DB consisting of cities with population >180K: (22) // based on example (18)

Save the new DF by using the method to_csv: (23)

BONUS:
loc() is a label-based data selecting method which means that we have to pass the name of the row or column that we want to select. **This method includes the last element of the range passed in it.**
Simple syntax for your understanding:
loc[row_label, column_label]

iloc() is an indexed-based selecting method which means that we have to pass an integer index in the method to select a specific row/column. **This method does NOT include the last element of the range passed in it.**
Simple syntax for your understanding:
iloc[row_index, column_index]

Access the value on row 1 and column 3: (24)

Access the column using the name: (25)

Let us create a new dataframe called 'df2' and assign 'df' to it. Now, let us set the "Name" column as an index column using the method set_index(): (26)

Valid statement: (27)

SLICING uses the [] operator to select a set of rows and/or columns from a DataFrame: (28)

So if you want to select (slice out) a set of rows i.e. 0, 1, and 2, your code would look like this: (29)

We can also select a specific data value using a row and column location within the DataFrame and iloc indexing. (30) // vs loc: (31)

```
(21)
dict = { 'c1': [1,1,2,2,3,2], 'c2': ['a', 'b', 'c', 'd', 'e', 'f']}
df = pd.DataFrame(dict);
print(df1['c1'].unique())

(22)
df2=df[df['Population'] > 180000]

(23)
df2.to_csv('result.csv')

(24)
df.iloc[0,2]

(25)
x = {'Name': ['Bell','Sandy', 'Jack', 'Peter'],
'ID': [1, 2, 3, 4], 'Department': ['D1', 'D2', 'D3', 'SE'],
'Salary':[200000, 90000, 70000, 80000]}
df = pd.DataFrame(x)  #casts dict to a DF
df.loc[0, 'Salary']

(26)
df2=df
df2=df2.set_index("Name")

(27) #Now, let us access the column using the name
df2.loc['Jack', 'Salary']

(28) data[start:stop]
Note: here the start represents the index from where to consider, and stop represents the index one step BEYOND the row you want to select. You can perform slicing using both the index and the name of the column.

(29) df.iloc[0:3]  #row 3 is not included, just rows 0-2

(30) df.iloc[0:2, 0:3]
# prints rows 0, 1 and columns 0, 1, 2

(31) print(df.loc[0:2,'ID':'Department'])
# prints rows 0, 1 AND 2; columns from 'ID' to 'Dept'
=> row 2 is included
```

4. PANDAS (II)

LOAD DATA WITH PANDAS

PANDAS = a popular library for data analysis
Import pandas using the import keyword (13)
Notes:
- This assumes the library is installed
- We now have access to a number of large pre-built classes and functions

Load a CSV file using panda's built-in function, read_csv: (14)
Note: df stands for "data frame" (you may choose another name, ofc)

You may use an alias instead of repeating the library name: (15)

Now that we have the data in a data frame, we can work with it. We can use the method **head** to examine the first 5 rows of a data frame: (16)
Note: The header will also be printed, in addition to the 5 rows

The process for reading an Excel file is similar: (17)

A DATA FRAME IS COMPRISED OF ROWS AND COLUMNS.
We can create a DATA FRAME out of a dictionary: (18)
Note: The keys correspond to the resulting table headers.
The values are lists corresponding to the rows.

Create a new DF consisting of one column: (19)
// or, multiple columns: (20)

```
(13) import pandas

(14) import pandas
csv_path = 'file1.csv'
df = pandas.read_csv(csv_path)
print(df)

(15) import pandas as pd
csv_path = 'file1.csv'
df = pd.read_csv(csv_path)
print(df)

(16) import pandas as pd
csv_path = 'file1.csv'
df = pd.read_csv(csv_path)
print(df.head())

(17) import pandas as pd
xlsx_path = 'file1.xlsx'
df = pd.read_excel(xlsx_path)
print(df.head())

(18) import pandas as pd
stats = {'Cities': ['Zuerich', 'Geneva', 'Basel'],
    'Population': [341000, 183000, 164000],
    'Strengths': ['financial hub, tourism, UZH', 'politics & diplomacy', 'university & museums']
    }
df = pd.DataFrame(stats)
print(df)

   Cities Population Strengths
0 Zuerich 341000 financial hub, tourism, UZH
1 Geneva 183000 politics & diplomacy
2 Basel 164000 university, museums

(19) x = df[['Cities']]     #double square brackets!
(20) y = df[['Cities', 'Population']]
```