Day 3: Programming with Python

File Input and Output (I/O)

Until now, we used the standard input and output to read and write. However, almost always we have our data in files and we want to read data from files and also write to them.

In this unit we will see basic functions and methods necessary to manipulate files in Python. Most of the file manipulation can be done with a file object.

0. Install and import

Let's start by importing all the required libraries. For this unit we need `csv` and `Pandas` in order to read and write to the files.

Pandas is part of the Anaconda distribution. If not installed, you can install it like:

```
conda install pandas
```

In the next unit we will explain all the basics for `Pandas`. For the moment we only import the libraries.

In [1]:
```
import csv
import pandas as pd
```

1. Reading and writing to a file

Key points:

- **open()** - Opens the file in a spcific mode
- **write()** - Writes a string to the file
- **read()** - Reads the file
- **readlines()** - Returns all lines in the file as list
- **readline()** - Returns the current line in the file
- **with open()** - When we use the statement "with" we do not have to close the file.

The first step for reading and writing a file is to open it with the `open()` function. `open()` takes as argument the path of the file and the mode to open it (e.g., `r` for reading, `w` for writing, `a` for appending).

Write to a file

Let's create a file and write 'Utrecht'

In [2]:
```
f = open("cities.txt", "w")
f.write('Utrecht')
f.close()
```

Let's create a list with some cities (e.g., ["Paris", "Rome", "Athens", "Utrecht", "Berlin"])

In [3]:
```
cities = ["Paris", "Rome", "Athens", "Utrecht", "Berlin"]
```

Let's try to write the cities to the file

In [4]:
```
f = open("cities.txt", "w")
f.write(cities)
f.close()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-f73b99f22eed> in <module>
      1 f = open("cities.txt", "w")
----> 2 f.write(cities)
      3 f.close()

TypeError: write() argument must be str, not list
```

The `write()` function takes as argument a string. We can iterate the cities list instead

In [5]:
```
cities = ["Paris", "Rome", "Athens", "Utrecht", "Berlin"]

f = open("cities.txt", "w")
for city in cities:
    f.write(city)
f.close()
```

If we open the file, we see that all the cities are written in the same line. Let's add a newline:

In [6]:
```
f = open("cities.txt", "w")

for city in cities:
    f.write(city + "\n")
f.close()
```

Read a file

To read a file we use again the `open()` function, this time in mode `r`

In [7]:
```
f = open("cities.txt", "r")
```

Now we use the function `read()` to read the contents. Let's create a variable `content` to store the data, `close()` the file and print the variable.

In [8]:
```
content = f.read()
```

In [9]:
```
f.close()
```

In [10]:
```
print(content)
```

```
Paris
Rome
Athens
Utrecht
```

```
Berlin
```

The `read()` function stores all the data into a variable and is not easy to work with every line. Instead we can read all the lines of a file with the `readlines()` function.

In [11]:
```
f = open("cities.txt", "r")
content = f.readlines()
print(content)
f.close()
```

```
['Paris\n', 'Rome\n', 'Athens\n', 'Utrecht\n', 'Berlin\n']
```

We also have the `readline()` function that reads only one line every time is called

In [12]:
```
f = open("cities.txt", "r")
content = f.readline()
print(content)
f.close()
```

```
Paris
```

In [13]:
```
f = open("cities.txt", "r")
content = f.readline()
print(content)

content = f.readline()
print(content)
f.close()
```

```
Paris

Rome
```

Until now we `open()` a file, store it in a variable, `read()` or `write()` to it and in the end we have to `close()` the file.

A more convenient way that does not require to `close()` the file in the end is using the `with` statement.

In [14]:
```
with open("cities.txt", "r") as f:
    for line in f.readlines():
        print(line, end="")
```

```
Paris
Rome
Athens
Utrecht
Berlin
```

Same happens with writing to a file

In [15]:
```
cities = ["Paris", "Rome", "Athens", "Utrecht", "Berlin"]

with open("cities.txt", "w") as f:
    for c in cities:
        f.write(c + '\n')
```

2. Reading and writing to CSV files

Key points:

- **csv.writer()** - Creates the writer object
- **csv_writer.writerow()** - Writes the data (one line)
- **csv.reader()** - The reader object

A common file that is used to store the data is the `CSV` file. CSV stands for Comma Separated Values and is a plain text file that stores tables and spreadsheet information. CSV files can be easily imported and exported using programs that store data in tables. We can open a CSV file with a text editor to view the data. An example can look like this:

```
country,capital,continent
Greece,Athens,Europe
France,Paris,Europe
Japan,Tokyo,Asia
Cuba,Havana,America
```

Let's see how to open and write csv files with Python. One way is to use the `csv` module that provides us functions to read and write.

We first open the file and then call the `csv.writer()` to write on it the list of the cities.

In [16]:
```
data = ["country,capital,continent",
        "Greece,Athens,Europe",
        "France,Paris,Europe",
        "Japan,Tokyo,Asia",
        "Cuba,Havana,America"
        ]

with open('cities.csv', 'w') as newFile:
    csv_writer = csv.writer(newFile)

    for line in data:
        csv_writer.writerow(line)
```

Let's try now to read the file and print the capitals.

In [17]:
```
with open('cities.csv') as f:
    csvReader = csv.reader(f)

    for line in csvReader:
        print(line[1])
```

```
o
r
r
a
u
```

This happened because `writerow()` expects a sequence of strings. Therefore we have to split the line to output the data in the right format.

In [18]:
```
with open('cities.csv', 'w') as f:
    csv_writer = csv.writer(f)
```

```
    for line in data:
        csv_writer.writerow(line.split(","))
```

And now read the file and print the capitals

In [19]:
```
with open('cities.csv') as f:
    csvReader = csv.reader(f)
    next(csvReader)
    for line in csvReader:
        print(line[1])
```

```
Athens
Paris
Tokyo
Havana
```

3. Reading and writing CSV files with Pandas

Key points:

- **pd.read_csv()** - Reads the csv file
- **df.to_csv()** - Writes the dataframe to csv

It is also possible to read different types of files with the help of other libraries such as `pandas`. We will cover `pandas` in the next unit. For now we will only see how we can use it to read and write csv files.

Let's read the `cities.csv` file.

In [20]:
```
df = pd.read_csv("cities.csv")
```

In [21]:
```
print(df)
```

```
  country capital continent
0  Greece  Athens    Europe
1  France   Paris    Europe
2   Japan   Tokyo      Asia
3    Cuba  Havana   America
```

The `read_csv()` function provides a variety pf different parameters, a description of which can be found here: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html. For example, if we set `header` parameter to `None` the following will happen.

In [22]:
```
df = pd.read_csv("cities.csv", header=None)
print(df)
```

```
         0        1          2
0  country  capital  continent
1   Greece   Athens     Europe
2   France    Paris     Europe
3    Japan    Tokyo       Asia
4     Cuba   Havana    America
```

Finally the `to_csv()` function exports the data to CSV format.

In [23]:
```
df.to_csv('../data/cities2.csv')
```

Exercises

1. During the unit, we created the cities.txt file. Read the file and print only line 3 from the file

In [24]:
```
cities = ["Paris", "Rome", "Athens", "Utrecht", "Berlin"]

with open("cities.txt", "w") as f:
    for c in cities:
        f.write(c + '\n')
```

In [25]:
```
with open("cities.txt", "r") as f:
    lines = f.readlines()
    print(lines[2])
```

```
Athens
```

2. Append to this file (cities.txt) two more cities (London and Florence)

In [26]:
```
cities = ['London', 'Florence']

with open("cities.txt", "a") as f:
    for city in cities:
        f.write(city + '\n')
```

3. Read the file and print its content. Print it in a way that every city is printed in a row without additional newlines

In [27]:
```
with open("cities.txt", "r") as f:
    lines = f.readlines()

for l in lines:
    print(l, end="")
```

```
Paris
Rome
Athens
Utrecht
Berlin
London
Florence
```

4. Read the cities.txt file and write its content to a new file (cities2.txt) after skipping line 2

In [28]:
```
count = 0
with open("cities.txt", "r") as f:
    lines = f.readlines()
with open("cities2.txt", "w") as f:
    for line in lines:
        if (count == 1):
            count += 1
            continue
```

```
        else:
            f.write(line)
        count+=1
```

5. Create a csv file called employers.csv (in the data folder) and write the data. Use as delimiter the symbol ;

In [29]:
```
data = ["name,department",
        "John,IT",
        "Anna,IT",
        "James,Management",
        "Peter,Accounting",
        "Luis,IT"
        ]

with open('../data/employers.csv', mode='w') as f:
    employee_writer = csv.writer(f, delimiter=';')

    for l in data:
        employee_writer.writerow(l.split(','))
```

6. Now read the employers.csv file and print only the names of those that are in IT

In [30]:
```
with open('../data/employers.csv', 'r') as f:
    reader = csv.reader(f, delimiter = ';')
    for row in reader:
        if row[1] == 'IT':
            print(row[0])
```

```
John
Anna
Luis
```

7. Read the employers.csv file again with Pandas and print it

In [31]:
```
df = pd.read_csv("../data/employers.csv", sep=";")
print(df)
```

```
    name  department
0   John          IT
1   Anna          IT
2  James  Management
3  Peter  Accounting
4   Luis          IT
```

In [ ]: