18-files

March 27, 2022

1 I/O and file processing

I/O stands for input/output, and it encompasses every operation a program makes with the outside world. Examples of such operations are: - printing or displaying something on the screen (the screen is outside the program) - reacting to mouse or keyboard commands (the user is outside the program) - sending information through the network (the network is outside the program) - opening, writing, and saving files (files are outside the program)

So far in the course, the programs you wrote were completely self-contained. Functions returned values depending on the parameters, and nothing more.

In this lecture we will see how you can write python programs that interact with the outside world.

1.1 Print

One of the simplest I/O operations is printing, which you are already familiar with.

[1]:
```
print("Hello world!")
```

```
Hello world!
```

Observe that the string `"Hello world!"` is printed on the screen, differently from:

[2]:
```
"Bye world!"
```

[2]:
```
'Bye world!'
```

which is shown on the screen for our convenience.

If this difference is hard to grasp, think of these two commands inside functions:

[3]:
```
def f1():
    print("This is printed, but the function returns nothing")

x = f1()
print("x =", x) # x is None, because the function returns nothing
```

```
This is printed, but the function returns nothing
x = None
```

[4]:
```
def f1():
    return "This is what the function returns."

x = f1()
print("x =", x)
```

```
x = This is what the function returns.
```

Printing can be quite useful, but it is very passive. It only allows the program to thrown some information into the outside world (*output*), but there is no way the outside world can influence the program. In order to achieve that, we need a way for the outside world to provide *inputs* to programs.

1.2 Input

One of the simplest ways of obtaining input from the outside world is to read it from the *standard input*. In practice, this means whatever is typed on the screen. This can be done using the python command `input()`.

When a program is running, and it encounters `input()`, it will stop and wait for the user to type something on the screen. After typing, the user presses `enter` to indicate to python that it can read what was typed, and continue the program. Python reads the input as a string, and this needs to be assigned to a variable.

[5]:
```
def isInputNumber():
    x = input()
    if x.isdigit():
        return True
    else:
        return False

isInputNumber()
```

```
Not a number
```

[5]:
```
False
```

[6]:
```
isInputNumber()
```

```
2022
```

[6]:
```
True
```

The `input()` command is useful for writing programs that interact with the user. Like the following one:

[7]:
```
def meetEliza():
    print("Hi! What is your name?")
    name = input()
    if name == "Eliza":
        print("What a coincidence! I am also called Eliza. Nice to meet you!")
    else:
        print("Hello " + name + "! Nice to meet you. My name is Eliza.")

meetEliza()
```

```
Hi! What is your name?
Rama
Hello Rama! Nice to meet you. My name is Eliza.
```

[8]:
```
meetEliza()
```

```
Hi! What is your name?
Eliza
What a coincidence! I am also called Eliza. Nice to meet you!
```

1.3 Reading files

Prints and inputs are good when we are dealing with small amounts of information, but when things become too large, files come to the rescue.

Python can open files using the `open(filename)` function. The parameter `filename` is the path of the file as a string. This function returns a `TextIOWrapper` object. You do not need to know what this object is, but you need to be aware that the `open(filename)` function will not return the content of the file as a string. To get the whole content of the file as a string, you need to use the `.read()` function (invoked using the dot notation).

If this file is in the same location as your program, it can be opened like this:

[9]:
```
file = open("sample.txt")
content = file.read()
print(content)
print(file) # A weird object.
```

```
This is the file sample.txt.
There is nothing interesting to see here… move on.

<_io.TextIOWrapper name='sample.txt' mode='r' encoding='UTF-8'>
```

If the file is somewhere else, you need to call `open` using the file *path*, which is the address of the file in your computer. For example, if this file is in your user folder, it could look something like this (for MacOS):

[10]:
```
# Will raise FileNotFoundError because this path does not exist on my computer
file = open("/Users/greis/sample.txt")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipykernel_1470565/1963991121.py in <module>
```

```
      1 # Will raise FileNotFoundError because this path does not exist on my␣
 ↪computer
----> 2 file = open("/Users/greis/sample.txt")

FileNotFoundError: [Errno 2] No such file or directory: '/Users/greis/sample.tx '
```

Many times it will be convenient to process a file line by line. In this case, we do not need to read the whole file at once into a variable using `.read()`. Instead, we can loop through the lines of a file like this:

[11]:
```python
file = open("numbers.txt")
s = 0
for line in file:
    n = int(line) # We know the line has one int
    s += n
print(s)
```

```
680
```

If we want to get all the lines in a list, we can use `.readlines()`:

[12]:
```python
file = open("numbers.txt")
lines = file.readlines()
print(lines)
```

```
['34\n', '65\n', '23\n', '87\n', '3\n', '65\n', '8\n', '14\n', '56\n', '73\n',
'69\n', '93\n', '4\n', '14\n', '27\n', '45\n']
```

1.3.1 CSV files

`csv` is a very popular file format for storing information. `csv` stands for *comma separated values*, and it is a file that contains values separated by commas. Each line corresponds to an entry, like a table row, and the values for that entry are separated by commas. So the table:

Movie	Year	Rating
The Grand Budapest Hotel	2014	88
Little Miss Sunshine	2006	91
The Darjeeling Limited	2007	67
Moonrise Kingdom	2012	84
Mary and Max	2009	81

corresponds to the file:

```
Movie,Year,Rating
The Grand Budapest Hotel,2014,88
Little Miss Sunshine,2006,91
The Darjeeling Limited,2007,67
Moonrise Kingdom,2012,84
```

```
Mary and Max,2009,81
```

In fact, any spreadsheet can be saved in this format.

Lines that start with # in `csv` files are comments and should be ignored (like in python).

Most of the datasets available for data analysis come in this format (see for example https://archive.ics.uci.edu/ml/index.php or https://www.kaggle.com/datasets), so it is useful to learn how to read and represent this kind of file.

If I ask you what is the rating of Moonrise Kingdom according to the table above, you will most likely find the Moonrise Kingdom line in the table, and then go to the "Rating" column. We would like to do the same in python, namely, if our dataset is stored in a variable `ds`, we would like to do:

```
ds["Moonrise Kingdom"]["Rating"]
```

The data structure that allows us to do that are dictionaries. Therefore, we will read the `csv` file as a dictionary of dictionaries. The outer dictionary is indexed by the rows (the movie titles), and its values are dictionaries. The inner dictionaries are indexed by the columns, and their values are the values on each position of the table.

This double dictionary is created by reading the file line by line, and, for each line we create the inner dictionary. Remember that the first line should not be processed.

One way of doing that is:

[13]:
```python
file = open("movies.csv")
lines = file.readlines()
# strip removes newline char, split splits the values at the commas
first_line_vals = lines[0].strip().split(",")
col_names = first_line_vals[1:]

ds = {}
for line in lines[1:]:
    vals = line.strip().split(",")
    line_name = vals[0]
    col_vals = vals[1:]
    inner_dict = {}
    for i in range(len(col_vals)):
        inner_dict[col_names[i]] = col_vals[i]
    ds[line_name] = inner_dict

ds
```

[13]:
```
{'The Grand Budapest Hotel': {'Year': '2014', 'Rating': '88'},
 'Little Miss Sunshine': {'Year': '2006', 'Rating': '91'},
 'The Darjeeling Limited': {'Year': '2007', 'Rating': '67'},
 'Moonrise Kingdom': {'Year': '2012', 'Rating': '84'},
 'Mary and Max': {'Year': '2009', 'Rating': '81'}}
```

Now we can get any value from the table using an intuitive indexing:

```
[14]: ds["Little Miss Sunshine"]["Year"]
```

```
[14]: '2006'
```

1.3.2 Reading files using the csv library

Since csv is a very popular format, python has its own library for handling this kind of file: https://docs.python.org/3/library/csv.html

We have different options to open a csv file using this library. The first one is using the `csv.reader` function, which takes as input a file (which was already opened using the `open` function), and returns a `reader` object. This reader object can be iterated to get each line of the file as a list of string.

```
[15]: import csv

file = open("movies.csv")
csv_reader = csv.reader(file)

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

```
['Movie', 'Year', 'Rating']
['The Grand Budapest Hotel', '2014', '88']
['Little Miss Sunshine', '2006', '91']
['The Darjeeling Limited', '2007', '67']
['Moonrise Kingdom', '2012', '84']
['Mary and Max', '2009', '81']
```

If the separators in the file are not commas, you can specify them with the optional `delimiter` parameter:

```
[16]: import csv

# Note that we could use whatever extension we want
file = open("movies-semi-colon.txt")
csv_reader = csv.reader(file, delimiter=';')

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

```
['Movie', 'Year', 'Rating']
['The Grand Budapest Hotel', '2014', '88']
['Little Miss Sunshine', '2006', '91']
['The Darjeeling Limited', '2007', '67']
['Moonrise Kingdom', '2012', '84']
['Mary and Max', '2009', '81']
```

Notice that we could have achieved the same thing without the `csv` library with a few extra lines of code:

[17]:
```
file = open("movies-semi-colon.txt")
lines = file.readlines()

for line in lines:
    lst = line.strip().split(";")
    print(lst)
```

```
['Movie', 'Year', 'Rating']
['The Grand Budapest Hotel', '2014', '88']
['Little Miss Sunshine', '2006', '91']
['The Darjeeling Limited', '2007', '67']
['Moonrise Kingdom', '2012', '84']
['Mary and Max', '2009', '81']
```

The `csv` library provides a function called `DictReader` that reads the file into dictionaries, but in a different way from what we did above. In this case, *each line becomes a dictionary* whose keys are the column names.

[18]:
```
file = open("movies.csv")
csv_dict_reader = csv.DictReader(file, delimiter=",")
print(csv_dict_reader.fieldnames)

for d in csv_dict_reader:
    print(d)
```

```
['Movie', 'Year', 'Rating']
{'Movie': 'The Grand Budapest Hotel', 'Year': '2014', 'Rating': '88'}
{'Movie': 'Little Miss Sunshine', 'Year': '2006', 'Rating': '91'}
{'Movie': 'The Darjeeling Limited', 'Year': '2007', 'Rating': '67'}
{'Movie': 'Moonrise Kingdom', 'Year': '2012', 'Rating': '84'}
{'Movie': 'Mary and Max', 'Year': '2009', 'Rating': '81'}
```

If the first line of the file is not the name of the columns, these can be specified by the optional `fieldnames` parameter (see this and other options at https://docs.python.org/3/library/csv.html#csv.DictReader).

1.4 Writing files

If we are computing something that we want to save, we can write the result into a file. To write into files, we first need to open it. But this time we need to indicate we are opening a file for writing. This is done by passing an extra parameter to the `open(filename)` function.

[19]:
```
file1 = open("results1.txt", "w") # w indicates the file is opened for writing
```

The `filename` parameter works as before: if you want to write to a file that is not in the same location as your python program, you need to write the path.

Attention: If the file does not exist, it will be created. If it exists, it will be *overwritten*.

If you want to write to a file that exists, but you do not want its content to be overwritten, you can open the file using `"a"`, for append:

[20]:
```
file2 = open("results2.txt", "a")
```

Once the file is open, we can write to it using the `.write(s)` function, which takes a string as a paramter.

[21]:
```
file1.write("Stuff to go into the file")
```

[21]:
```
25
```

This function returns the number of characters written.

1.5 Exercise

The file https://web2.qatar.cmu.edu/cs/15110/resources/zoo.csv is a dataset about animals and their characteristics. Download and open it in a text editor (notes or notepad) to inspect it and find out what are the column names. For this file in particular, they are not given on the first line.

Implement the function `readZoo()` that reads the `zoo.csv` file and returns a dictionary of dictionaries corresponding to this dataset.

Once you have this dictionary, write functions that take it as input and return: - all mammals that lay eggs - all mammals that are aquatic - which animals have five legs

[22]:
```
def readZoo():
    return {}
```