4. Working with Data in Python

Overview

1. Python application of the week
2. Recap of last time
3. Working with files
4. Data analysis with Pandas

Example Application: Drug Discovery

Recap: Conditions

- An essential part of all programming languages is the ability to run pieces of code *if* a condition is true.
- In Python this is done with the `if-else` statement.

```
temp = 68
if temp > 100:
    print('the water will boil')
    else:
        print('the water will not boil')
```

- The colon, along with the indentation, is used to separate each logical block of code.

Recap: Loops

- There are many situations in programming when it is useful to perform some action multiple times.
- This can be achieved in Python with either `for` loops:

```
shopping_list = ['milk', 'eggs', 'rice', 'apples']

for item in shopping_list:
    print(item)
```

- Or `while` loops:

```
x = 0
while x < 5:
    print(x)
    x = x + 1
```

Recap: Functions

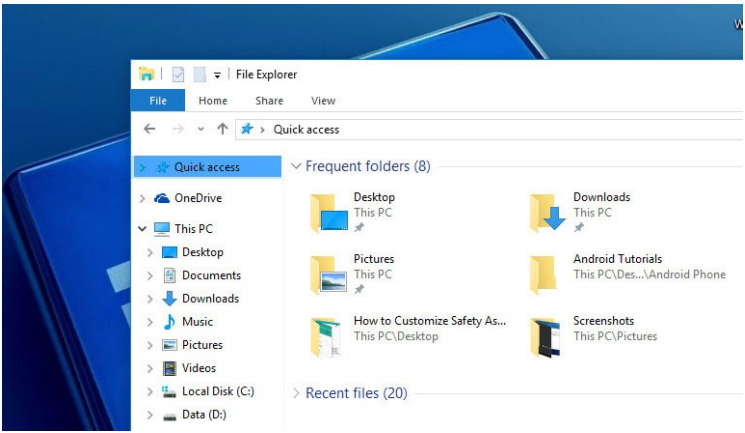
- Functions are small, reusable blocks of code that can take inputs and return outputs.
- Some functions are built in to Python, like `len()` and `type()`.
- You can define your own function using the `def` key-word.

```
def cel_to_fah(C):
    F = 1.8 * C + 32
    return F

temp_cel = 25
temp_fah = cel_to_fah(temp_cel)
print(temp_fah)
```

```
Out: 77.0
```

New topic: working with files



Computer files: an overview

- Computer files can broadly be split into two categories:

    1. Plain text files.
    2. Files with a special encoding.

- Plain text files are relatively more simple. They are made up of regular characters with no font, text size, color etc and can be opened with a text editor (e.g. *Notepad* on Windows, *TextEdit* on Mac or *Gedit* on Linux). Examples include `.txt` and `.csv` files.
- Files with a special encoding need to be opened with a special programme, e.g. Microsoft word. Their raw data is expressed in a specific way only interpretable by certain programmes.

Computer files: an overview

Plain text	Use	Special Encoding	Use
`.txt`	Pure text	`.docx`	MS word
`.csv`	Table data	`.xlsx`	MS excel
`.py`	Python code	`.mp3`	Compressed audio
`.html`	Website display code	`.pdf`	Final format documents
`.json`	Structured data	`.jpeg`	Compressed images
`.ipynb`	Jupyter notebook (really just json)	`.exe`	Windows executable

`.txt` files

- These are the simplest file, just containing text intended to be read normally.

```
Lorem ipsum enim orci tellus augue nam vitae, conubia dui placerat ut
condimentum mollis conubia semper, mi habitant mollis pretium commodo
platea quis vel facilisis inceptos.

Volutpat placerat scelerisque mauris sapien nisl ipsum tempus taciti
fusce iaculis vitae tempus, sapien proin ullamcorper ipsum id consequat
inceptos per himenaeos mauris id vitae eros fringilla phasellus
fermentum aliquet aliquam bibendum fringilla commodo.

Ligula enim cursus rutrum eu ante feugiat magna quisque sociosqu
bibendum, sit dolor accumsan velit congue augue elementum consequat in,
ligula facilisis platea ut odio platea suscipit adipiscing a.
```

`.csv` files

- Comma Separated Value files are simple spreadsheets and can be opened or saved in excel.
- The first row is often a header, with the following rows containing data, with each column separated by a comma.

```
Air Quality Index,Longitude,Latitude
28,120.409653,23.925175
42,121.493806,25.072611
25,120.758833,24.382942
42,121.451861,24.982528
33,121.515389,25.105417
37,121.513311,25.0632
28,120.677689,24.099611
28,121.201811,25.060344
44,120.425081,22.565747
```

`.html` files

- HTML (Hyper-Text Markup Language) files are plain text files that tell a web browser what to display on your screen.

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>The HTML5 Herald</title>
</head>

<body>
  The content goes here
  <script src="js/scripts.js"></script>
</body>

</html>
```

`.json` files

- JSON files are used to store structured data.

```
{
  "data": {
      "type": "articles",
      "id": 1,
      "attributes": {
          "title": "JSON:API paints my bikeshed!",
          "body": "The shortest article. Ever.",
          "created": "2015-05-22T14:56:29.000Z",
          "updated": "2015-05-22T14:56:28.000Z"
                    }
          }
}
```

Opening plain text files in Python

- Python by itself cannot be used for most files with a special encoding. You may be able to handle them in Python, but will often need to install special libraries. Plain text files however can be dealt with in Python directly.
- Say we have a file called `sample.txt` with the following contents:

```
Lorem ipsum enim orci tellus augue nam vitae, conubia dui placerat ut condimentum mollis conubia semper, mi habitant mollis pretium commodo platea quis vel facilisis inceptos.

Volutpat placerat scelerisque mauris sapien nisl ipsum tempus taciti fusce iaculis vitae tempus, sapien proin ullamcorper ipsum id consequat inceptos per himenaeos mauris id vitae eros fringilla phasellus fermentum aliquet aliquam bibendum fringilla commodo.
```

Opening plain text files in Python

- We can open this file using the built in `open()` function.
- Once the file is opened, the contents can be read and saved as a string.

```
# open a file
file_object = open('sample.txt')

# save the file content as a string
data = file_object.read()

# print file content
print(data)

#close the file
file_object.close()
```

Opening plain text files in Python

- The file should be closed after usage (but the world probably won't end if you don't). This can be done with the `file_object.close()` command.

- The file `sample.txt` must be contained in the same folder where you are running your python. Otherwise the following error will arise:

```
FileNotFoundError: [Errno 2] No such file or directory: 'sample.txt'
```

Opening plain text files in Python

- A good habit is to use a special piece of syntax called the `with` statement.

- This does the same as above, but automatically closes the file at the end.

```
# using "with statement" with open() function
with open('sample.txt') as file_object:

    # read file content
    data = file_object.read()

    # print file contents
    print(data)
```

Example: parsing a `.csv` file

```
Air Quality Index,Longitude,Latitude
28,120.409653,23.925175
42,121.493806,25.072611
25,120.758833,24.382942
42,121.451861,24.982528
33,121.515389,25.105417
37,121.513311,25.0632
28,120.677689,24.099611
28,121.201811,25.060344
44,120.425081,22.565747
32,120.337736,22.565833
31,121.526528,25.062361
42,121.221667,24.953278
```

Example: parsing a `.csv` file

```
# create empty list to hold the data
data = []

# open using "with statement" and open() function
with open('air_quality.csv') as csv_file:

    # loop through each line
    lines = csv_file.readlines()
    for line in lines[1:]:

        # turn string into list of strings
        numbers = line.split(',')
        data.append(numbers)

print(data)
```

Example: parsing a `.csv` file

```
[['28', '120.409653', '23.925175'],
 ['42', '121.493806', '25.072611'],
 ['25', '120.758833', '24.382942'],
 ['42', '121.451861', '24.982528'],
 ['33', '121.515389', '25.105417'],
 ['37', '121.513311', '25.063276']]
```

- Note that, because we read the contents of this CSV in from a file, and saved those contents as a string, all of these numbers are still in string form. If we want to use them as numbers, we'd have to explicitly go through each one and convert them into an `int` or a `float`.

Batch-processing many files

- Say we had a folder containing many files we would like to open and process. We can get the names of all these files using the function `os.listdir()`.
- `os.listdir(folder)` returns a list of all the filenames in `folder`.
- `os` is a *module* and must be imported.

```
import os

folder = 'C:/Users/Ed/Documents/my_folder'
file_names = os.listdir(folder)
print(file_names)
```

```
['file1.txt', 'file2.txt', 'file3.txt']
```

Batch-processing many files

- Given this list of file names, we are now able to loop over each name and open the contents.

```
# initialise an empty list to hold the contents of the files
all_files_content = []

# use a for loop to go through each file name
for file_name in file_names:

    # create a string holding the full path to the file
    full_path = folder + '/' + file_name

    # open the file
    with open(full_path) as file:

        # add the file contents to our list
        all_files_content.append(file.read())
```

Writing files

- As well as reading files already saved on your computer, Python can also be used to write new files.
- This is also achieved with the `open()` function, in write mode. To tell Python we want to *write* a file, we have to pass the letter `'w'` as a string to the `open()` function after the file name.

```
my_text = 'This is some text that I would like to save to a file'

# use the open function in write mode ('w' !!)
with open('my_text_file.txt', 'w') as my_file:

    # write the string to the file
    my_file.write(my_text)
```

- The file `my_text_file.txt` will now be saved into the current folder.

Writing files

- To write content on a new line, make sure the string passed to `.write()` ends with a new line character (`\n`).

```
title = 'A Blog About Prawn Cocktail Crisps'
date = '29-10-2020'
content = """I really love prawn cocktail crisps. They are salty and
sweet just like me. Cheese and onion? Please. Salt and vinegar? You must
be kidding."""

with open('blog_post.txt', 'w') as blog_post:

    blog_post.write(title + '\n')
    blog_post.write(date + '\n')
    blog_post.write(content + '\n')
```

Writing files

`blog_post.txt`:

```
A Blog About Prawn Cocktail Crisps
29-10-2020
I really love prawn cocktail crisps. They are salty and sweet just like
me. Cheese and onion? Please. Salt and vinegar? You must be kidding.
```

Writing files

- You can also write multiple lines using `.writelines()`.
- This function expects a list of strings as its argument.

```
lines = ['one \n', 'two \n', 'three \n']

with open('numbers.txt', 'w') as number_file:

    number_file.writelines(lines)
```

```
One
Two
Three
```

New topic: data analysis with Pandas

Pandas

- Pandas is a Python library that is used for data stored in table form.
- It's kind of like a super-charged excel, with all the extra power and flexibility that Python brings.
- The central object pandas provides is called a *DataFrame*. This is essentially a table.

Date	**Temp (C)**	**Rainfall (mm)**
01-01-2020	7	8
02-01-2020	6	2
03-01-2020	8	15
...	...	...

Pandas

- There are many ways to create a new DataFrame. The first is by specifying the data directly in a list.

```
# pandas is a library so must be imported
import pandas as pd

# a list of row data
data = [['01-01-2020', 7, 8],
        ['02-01-2020', 6, 2],
        ['03-01-2020', 8, 15]]

# specify the column names
column_names = ['Date', 'Temp (C)', 'Rainfall (mm)']

# create and print the dataframe
my_dataframe = pd.DataFrame(data, columns=column_names)
print(my_dataframe)
```

Pandas

Output:

	Date	Temp (C)	Rainfall (mm)
0	01-01-2020	7	8
1	02-01-2020	6	2
2	03-01-2020	8	15

- A pandas DataFrame is it's own specific *type*, just like lists, strings etc.

```
print(type(my_dataframe))
```

```
pandas.DataFrame
```

Pandas: columns

- We can extract individual columns from the DataFrame using the same bracket notation used for list and string indexing.
- Instead of passing a number, you can pass the column name

```
my_dataframe['Temp (C)']
```

	Temp (C)
0	7
1	6
2	8

Pandas: columns

- You can also add a new column in a similar way.

```
my_dataframe['Wind Speed (mph)'] = [2, 4, 5]
my_dataframe
```

	Date	Temp (C)	Rainfall (mm)	Wind Speed (mph)
0	01-01-2020	7	8	2
1	02-01-2020	6	2	4
2	03-01-2020	8	15	5

Pandas: columns

- A column can be removed in a number of ways. One is to use the `del` keyword.

```
del my_dataframe['Wind Speed (mph)']
```

	Date	Temp (C)	Rainfall (mm)
0	01-01-2020	7	8
1	02-01-2020	6	2
2	03-01-2020	8	15

Pandas: columns

- If you multiply, divide, add to or subtract from a numeric column, the operation is applied to the whole column.

```
my_dataframe['Temp (F)'] = 1.8 * my_dataframe['Temp (C)'] + 32
```

	Date	Temp (C)	Rainfall (mm)	Temp (F)
0	01-01-2020	7	8	44.6
1	02-01-2020	6	2	42.5
2	03-01-2020	8	15	46.4

Pandas: the index column

- In Pandas it's possible to have a special column called the index. This can be thought of as giving a name to each row.
- By default, the index is just the numbers `[0, 1, 2, ...]`, but in this example it may make sense to make the date column the index.

```
my_dataframe = my_dataframe.set_index('Date')
```

Date	Temp (C)	Rainfall (mm)	Temp (F)
01-01-2020	7	8	44.6
02-01-2020	6	2	42.5
03-01-2020	8	15	46.4

Pandas: the index column

- The index column can be used to select individual rows.
- The is done again with square brackets, but using `.loc[]`.

```
my_dataframe.loc['01-01-2020']
```

	01-01-2020
Temp (C)	7
Rainfall (mm)	8
Temp (F)	44.6

Pandas: the index column

- In fact, `.loc[]` can be passed both an index and a column, to select an individual item of data.

```
data_point = my_dataframe.loc['01-01-2020', 'Temp (C)']
print(data_point)
```

```
7
```

Reading data from a CSV file

- One common way to produce a pandas DataFrame is to read it in directly from a CSV file. Let's assume we have a CSV file called `air_quality.csv` that looks something like this:

```
ID,Air Quality,Longitude,Latitude
AA2013,28,120.409653,23.925175
AB3342,42,121.493806,25.072611
AD0293,25,120.758833,24.382942
AG5032,42,121.451861,24.982528
```

- Assuming this file is in the current working directory, we can read it in as

```
df = pd.read_csv('air_quality.csv')
```

Reading data from a CSV file

- If we want to designate one column as the index column, this can be done in the `read_csv` function directly.

```
df = pd.read_csv('air_quality.csv', index_col='ID')
print(df)
```

ID	Air Quality	Longitude	Latitude
AA2013	28	120.409653	23.925175
AB3342	42	121.493806	25.072611
AD0293	25	120.758833	24.382942
AG5032	42	121.451861	24.982528

Pandas: final thoughts

- Pandas is a large and powerful library. We've only scratched the surface of what you can do with it.
- The best way to learn it is just to start using it for your particular data analysis project, and just google every time you don't know what to do next - someone is almost guaranteed to have had the same issue.
- For more information on what you can do with Pandas, check out the documentation here.

DataKirk

Thanks!