Python Introduction
Part 3

Aslak Johansen asjo@mmmi.sdu.dk

May 6, 2026

prosa

Part 1:
HTTP Client Operations

HTTP Client Operations ▷ Client Module

We will be using the `requests` module.

Install:

```
pip install requests
```

HTTP Client Operations ▷ Client Module

We will be using the `requests` module.

Install:

```
pip install requests
```

Note: There are lots of other options for doing HTTP operations.

HTTP Client Operations ▷ GET Operations

```
import requests

response = requests.get("https://www.prosa.dk")

print(response.status_code)
print(response.text)
```

HTTP Client Operations ▷ GET Operations

```
import requests

response = requests.get("https://www.prosa.dk")

print(response.status_code)
print(response.text)
```

Similar functionality exists for POST, PUT ...

HTTP Client Operations ▷ GET Operations

```
import requests

response = requests.get("https://www.prosa.dk")

print(response.status_code)
print(response.text)
```

Similar functionality exists for POST, PUT ...

Documentation: https://requests.readthedocs.io/en/stable/

Part 2:
XLSX Files

XLSX Files ▷ Loading and Storing

```
import openpyxl

# one of
wb = openpyxl.Workbook()
wb = openpyxl.load_workbook(filename='input.xlsx')

# work with wb ...

wb.save("output.xlsx")
```

XLSX Files ▷ Sheets

```
sheet = wb.active

sheets = wb.worksheets
first_sheet = sheets[0]

first_sheet.title = "new title"
```

XLSX Files ▷ Cells

```
>>> from openpyxl.utils import get_column_letter
>>> sheet = wb.active
>>> cell = sheet["A1"]
>>> cell.value = 1
>>> print(sheet.cell(row=1, column=1).value)
1
>>> get_column_letter(1)
'A'
```

XLSX Files ▷ Fonts

```
>>> from openpyxl.styles import Font
>>> help(Font)

>>> font = Font(bold=True, italic=True)
>>> cell.font = font
```

XLSX Files ▷ Fonts

```
>>> from openpyxl.styles import Font
>>> help(Font)
```

```
>>> font = Font(bold=True, italic=True)
>>> cell.font = font
```

Documentation:
https://openpyxl.readthedocs.io/en/stable/api/openpyxl.styles.fonts.html#module-openpyxl.styles.fonts

XLSX Files ▷ Horizontal Alignment

```
>>> from openpyxl.styles import Alignment
>>> center = Alignment(horizontal='center')
>>> cell.alignment = center
```

XLSX Files ▷ Background Color

```
>>> from openpyxl.styles import PatternFill
>>> red_fill = PatternFill(start_color="FF0000", fill_type="solid")
>>> cell.fill = red_fill
```

XLSX Files ▷ Merging Cells

```
>>> sheet.merge_cells('A1:D1')
>>> sheet.merge_cells(start_row=3, start_column=1, end_row=3, end_column=4)
```

XLSX Files ▷ Formulas

```
>>> sheet["A1"].value = 1.2
>>> sheet["A2"].value = 1.8
>>> sheet["A3"].value = "=A1+A2"
```

Part 3:
Next Steps

SymPy

```
import sympy

s = input('Please enter an expression over i and j: ')
expr = sympy.sympify(s)
symi = sympy.symbols('i')
symj = sympy.symbols('j')

print('expr='+str(expr))
print('diff(expr, i)='+str(sympy.diff(expr, symi)))

for i in range(10):
    j = i % 3
    v = expr.subs([(symi, i), (symj, j)])
    print('i=%d j=%d => result=%f' % (i, j, v))
```

SymPy

```
import sympy

s = input('Please enter an expression over i and j: ')
expr = sympy.sympify(s)
symi = sympy.symbols('i')
symj = sympy.symbols('j')

print('expr='+str(expr))
print('diff(expr, i)='+str(sympy.diff(expr, symi)))

for i in range(10):
    j = i % 3
    v = expr.subs([(symi, i), (symj, j)])
    print('i=%d j=%d => result=%f' % (i, j, v))
```

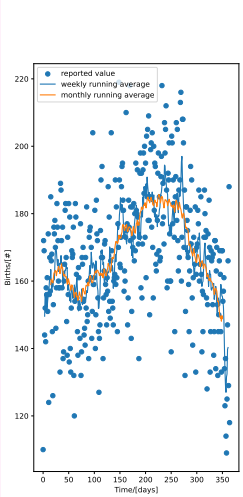
```
Please enter an expression over i and j: 3*i*i+4.83*i+j
expr=3*i**2 + 4.83*i + j
diff(expr, i)=6*i + 4.83
i=0 j=0 => result=0.000000
i=1 j=1 => result=8.830000
i=2 j=2 => result=23.660000
i=3 j=0 => result=41.490000
i=4 j=1 => result=68.320000
i=5 j=2 => result=101.150000
i=6 j=0 => result=136.980000
i=7 j=1 => result=181.810000
i=8 j=2 => result=232.640000
i=9 j=0 => result=286.470000
```

Pandas

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'raw'])
df['avgw'] = df['raw'].rolling(window=7, center=True).mean()
df['avgm'] = df['raw'].rolling(window=30, center=True).mean()

plt.figure(figsize=[5,10])
plt.xlabel('Time/[days]')
plt.ylabel('Births/[#]')
plt.scatter(df['day'], df['raw'],label='reported value')
plt.plot(df['day'], df['avgw'],label='weekly running average')
plt.plot(df['day'], df['avgm'],label='monthly running average')
plt.legend(loc=2)
plt.savefig('running-average.pdf')
```



AsyncIO

```
import asyncio
from aiohttp import web

async def handler (request):
    method  =         request.method
    path    = '/'+str(request.rel_url)[1:]
    payload =   await request.content.read()

    try:
        with open(path) as fo:
            return web.Response(status=200, text=''.join(fo.readlines()))
    except:
        return web.Response(status=500, text=str([method, path]))

async def main(interface, port):
    proto  = web.Server(handler)
    server = await loop.create_server(proto, interface, port)

loop = asyncio.get_event_loop()
asyncio.Task(main('0.0.0.0', 8080))

# enter service loop
try:
    loop.run_forever()
except KeyboardInterrupt:
    print('')
    print('STATUS: Exiting ...')
    loop.close()
    exit(0)
```

AsyncIO

```
import asyncio
from aiohttp import web

async def handler (request):
    method  =         request.method
    path    = '/'+str(request.rel_url)[1:]
    payload =   await request.content.read()

    try:
        with open(path) as fo:
            return web.Response(status=200, text=''.join(fo.readlines()))
    except:
        return web.Response(status=500, text=str([method, path]))

async def main(interface, port):
    proto  = web.Server(handler)
    server = await loop.create_server(proto, interface, port)

loop = asyncio.get_event_loop()
asyncio.Task(main('0.0.0.0', 8080))

# enter service loop
try:
    loop.run_forever()
except KeyboardInterrupt:
    print('')
    print('STATUS: Exiting ...')
    loop.close()
    exit(0)



$ curl localhost:8080/etc/hostname
thera
```

Cairo

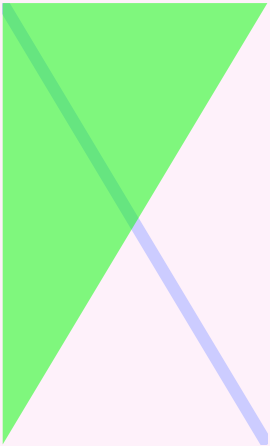
```
import cairo

mm = lambda value: float(value)/25.4*72
width, height = mm(60), mm(100)

surface = cairo.PDFSurface('cairoexample.pdf', width, height)
c = cairo.Context(surface)

c.move_to(0, 0)
c.line_to(width, height)
c.set_line_width(7.8)
c.set_source_rgb(0.8, 0.8, 1)
c.stroke()

c.move_to(0,0)
c.line_to(width,0)
c.line_to(0,height)
c.close_path()
c.set_source_rgba(0, 1, 0, 0.5)
c.fill()
```



Other Resources

- This material: https://python.asjo.dk
- Official Python Tutorial: https://docs.python.org/3/tutorial/index.html
- *Think Python 2nd Edition* by Allen B. Downey (free PDF) https://greenteapress.com/wp/think-python-2e/

Part 4:
Exercises

Exercises ▷ Multiplication Table

Steps:

1. From Python, produce an XLSX file that has one sheet.
2. This sheet contains the 10s table.
   That is, a table that contains all products of combinations of integers between 1 and 10.
3. The cells containing the two factors should be in bold.
   That is, the top row and the left-most column.
4. the result cells should be defined through a formula over the factor cells.
5. The diagonal should have a blue background. These are the square numbers.

Exercises ▷ Download Data

Go to the UN data portal (https://data.un.org).

Under *"Popular statistical tables, country (area) and regional profiles"*, find the *"Population, surface area and density"* dataset.

Copy the link to the CSV version of this dataset (via right-click menu).

Make a script that downloads this link and stores the resulting data in a local file as `popdata.csv`.

Exercises ▷ Process Data

Make a new script that:

1. Reads `popdata.csv` from the last exercise.
2. Process the data:
   - 2.1 The second column contains an area descriptor. Drop all rows that do not have a country name in this row.
   - 2.2 The fourth column is the metric. Drop all rows that do not cover the "Population density" metric.
   - 2.3 The third column is the year. Drop all but the newest year that has the "Population density" metric.
   - 2.4 The fifth column has the value (i.e., the population density). We care about this.
3. Based on the processed data, produce a data structure that maps country name to population density.
4. Marshals the data structure to JSON format and store it as `popdata.json`.

Exercises ▷ Countries and Continents [1/3]

A python script is shown in the following two slides.

What does it do?

How does the mechanism for stepping through the input data work?

What is the purpose of the call to `lower()`?

Exercises ▷ Countries and Continents [2/3]

```
#!/usr/bin/env python

from sys import exit
import requests
import re
import json

pattern_continent = re.compile("^<h2 id=\"([^\"]+)\">")
pattern_country   = re.compile("^wiki/Flag_of_([^\"]+)\"")

# make wikipedia robot policy happy:
headers = {
  "User-Agent": "Demo for python requests module https://python.asjo.dk"
}

url = "https://simple.wikipedia.org/wiki/List_of_countries_by_continents"
response = requests.get(url, headers=headers)

if not response.status_code==200:
  print("Error fetching url: "+response.text)
print("Downloaded. Processing will take some time (90s on my machine) ...")
```

Exercises ▷ Countries and Continents [3/3]

```
# process
contents = response.text
continent = None
mapping = {}
while len(contents)>0:
  mo_continent = pattern_continent.match(contents)
  mo_country = pattern_country.match(contents)

  if mo_continent:
    continent = mo_continent.group(1).replace("_", " ")
    contents = contents[len(mo_continent.group(0)):]
  if mo_country:
    country = mo_country.group(1).replace("_", " ").lower()
    if not "&quot" in country:
      mapping[country] = continent
    contents = contents[len(mo_country.group(0)):]
  else:
    contents = contents[1:]

# store
s = json.dumps(mapping, sort_keys=True, indent=4, separators=(',', ': '))
with open("country2continent.json", "w") as fo:
  fo.writelines([s])
```

Exercises ▷ Putting It All Together

Note: This exercise builds on the last three exercises.

The exercise is to produce an XLSX file with a sheet per continent. Each sheet should contain a list of countries that belong to this continent (and their population densities).

They should be listed in order of population density, and the background color should act as a heatmap.

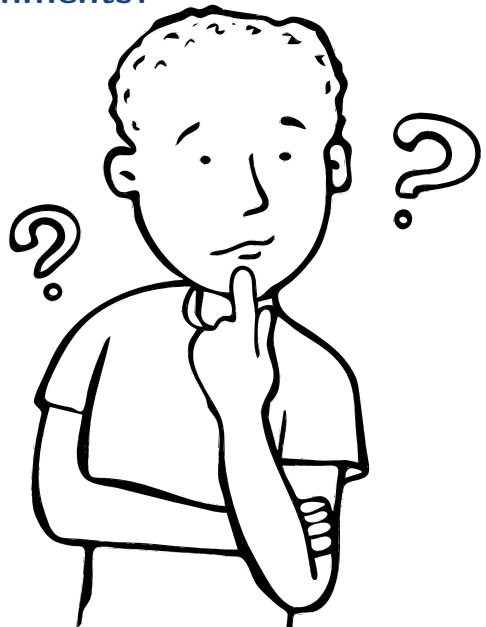
That is, the country with the highest population density (of the continent) should have a red background, and the one with the lowest should have a white background. All else should fall somewhere in between.

Hint: An integer can be converted to a double-digit hex value: `"%02x" % value`

Where to Find

https://python.asjo.dk

Questions and Comments?



https://openclipart.org/detail/238687/boy-thinking-of-question