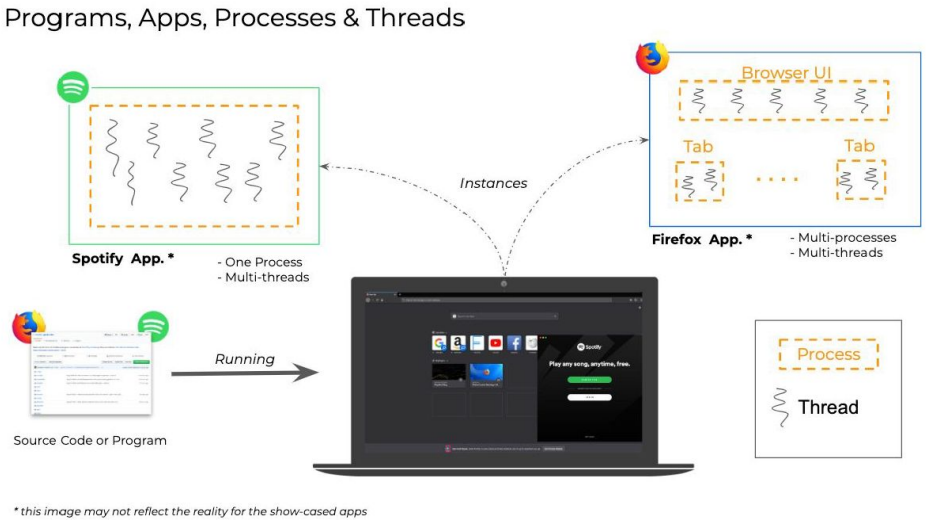
Data Science Survival Skills

Homework 9

Description of the Homework

In this week's homework, we will be looking at how to make your code faster.



Homework 9: Tasks 1/4

- This task is about code profiling. Use line-by-line profiling (“%lprun” command) to find the bottleneck in the following function.

```
import numpy as np
from skimage import data, color
from skimage.transform import resize

imgs = np.uint8(data.lfw_subset()*255)

def res_skimage(imgs):
    new_size = (imgs[1].shape[0]//2, imgs[1].shape[1]//2)
    res_im = []
    for im in imgs:
        image_resized = resize(im, new_size, anti_aliasing=True)
        res_im.append(image_resized)
    return np.asarray(res_im)
```

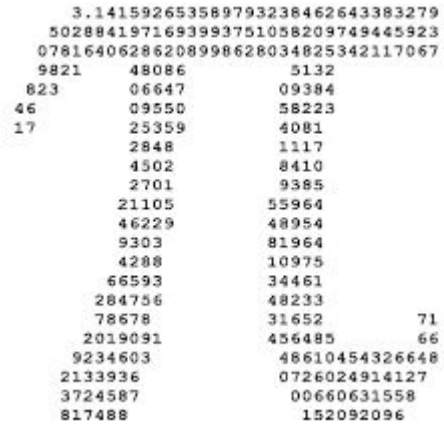
Homework 9: Tasks 1/4

- Once you have identified the inefficient part of the function, find a way to improve the speed of the code.
- Perform line-by-line profiling again on the updated function.

➜ **Slide:** Screenshot of the results of your line-by-line profile of the corrected function, with the bottleneck identified and the update marked by you so that we can see that you have found it (e.g. with a red arrow pointing to a specific line).

Homework 9: Task 2/4

- We will provide you with a function that approximates Pi (next slide). The approximation becomes more accurate as the number of iterations N is increased. We have prepared a list of different numbers that represent different values for N. Calculate an approximation for Pi for each value N. Parallelize the execution. Therefore use multithreading or multiprocessing (decide which one is better suited!).



Homework 9: Task 2/4

```python
def approximate_pi(n):
    pi_2 = 1
    nom, den = 2.0, 1.0
    for i in range(n):
        pi_2 *= nom / den
        if i % 2:
            nom += 2
        else:
            den += 2
    return 2*pi_2

# Data to pass to the above function
nums = [1_822_725, 22_059_421, 32_374_695,
        88_754_320, 97_162_66, 200_745_654]
```

➜ **Slide:** Explanation why you decided to use multithreading or multiprocessing.

➜ **Slide:** Total execution times for sequential and parallel processing of the different “nums”. How much faster could you make your code by just using multithreading/multiprocessing?

Homework 9: Task 3/4

- By using multithreading or multiprocessing for the function calls, we have not increased the speed within the function. Increase the execution speed by using numba and/or Cython!
- Note: It can be advantageous to stop using multithreading/multiprocessing, as the creation of a new process can be slower than the sequential call of the (numba-/cython-) optimized function.

➜ **Slide:** Screenshot of the code you used to optimize your function for execution speed.

➜ **Slide:** A written statement of how much faster you are (one complete sentence).

Homework 9: Task 4/4

- Of course we are also interested in the results.

➜ **Slide:** Plot of your Pi estimations with the correct value of Pi as a horizontal line.

Homework 9: Example solution

Task 1

```
Total time: 0.134479 s
File: /tmp/ipykernel_11691/2918951301.py
Function: res_skimage at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def res_skimage(imgs):
     2                                               new_size = (imgs[1].shape[0]//2, imgs[1].shape[1]//2)
     3                                               res_im = []
     4                                               for im in imgs:
     5                                                   image_resized = resize(im, new_size, anti_aliasing=True)
     6                                                   res_im.append(image_resized)
     7                                               return np.asarray(res_im)
```

This is the line that was the bottleneck. The highlighted line has been updated so that the bottleneck is now minimized.

Task 2

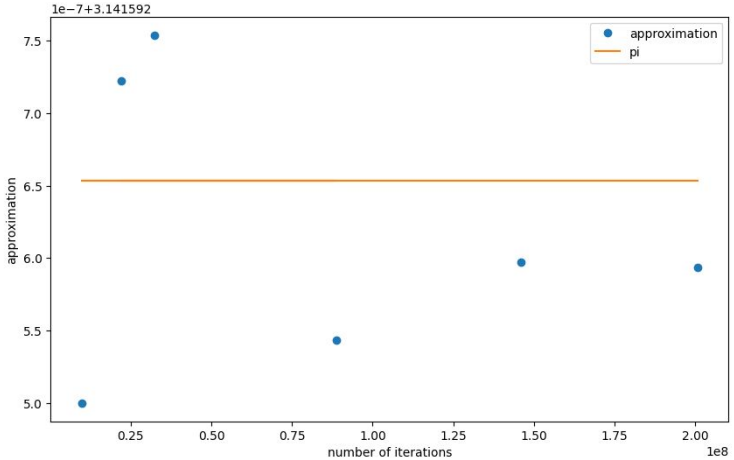
I used multithreading because this task is perfect for it (multiprocessing wouldn't work in this case because...). The sequential call of the function for each value of N took a total of 42 minutes. By using multithreading, I was able to reduce this time to 20 minutes. That is a huge reduction of 52%.

Task 3

```python
@PleaseBeFasterDecorator
def approximate_pi(n):
    pi_2 = 1
    nom, den = 2.0, 1.0
    for i in range(n):
        pi_2 *= nom / den
        if i % 2:
            nom += 2
        else:
            den += 2
    return 2*pi_2
```

By using the PleaseBeFasterDecorator I am over 9000% faster.

Task 4



Homework: Requirements

You must complete **all** homework assignments (**unless otherwise specified**) following these guidelines:

- **One** slide/page.
- **PDF** file format only.
- It has to contain your **name, student** (**matriculation**) **number** and **IdM** in the down-left corner.
- Font: **Arial,** Font-size: **> 10 Pt.**
- Answer **all** the questions and solve all the tasks requested.
- Be careful with **plagiarism**. Repeated solutions will not be accepted!