

Introduction to programming

Valentina Giunchiglia and Dragos Gruia

Pre-material

<https://github.com/dragos-gruia/MSc-Neuroscience-Python-Course-Development>

📁 Data	final	14 days ago
📁 Exercises	Update Exercises.md	14 days ago
📁 Figures	exercise	19 days ago
📁 __pycache__	remove cell outputs	12 days ago
📄 .gitignore	exercise	19 days ago
📄 Lecture1.ipynb	remove cell outputs	12 days ago
📄 Lecture10.ipynb	remove cell outputs	12 days ago
📄 Lecture2.ipynb	remove cell outputs	12 days ago
📄 Lecture3.ipynb	remove cell outputs	12 days ago
📄 Lecture4.ipynb	remove cell outputs	12 days ago
📄 Lecture5.ipynb	remove cell outputs	12 days ago
📄 Lecture6.ipynb	remove cell outputs	12 days ago
📄 Lecture7.ipynb	remove cell outputs	12 days ago
📄 Lecture8.ipynb	remove cell outputs	12 days ago
📄 Lecture9.ipynb	remove cell outputs	12 days ago
📄 README.md	Fix typos	12 days ago
📄 Setup-Mac.md	finals	16 days ago
📄 Setup-Windows.md	Update Setup-Windows.md	11 days ago
📄 rank_inv.py	Merge branch 'main' of https://github.com/dragos-gruia/MSc-Neuroscience-Python-Course-Development	19 days ago



We expect you to have completed **ALL** the lectures

Pre-material

If you didn't complete all the exercises, it's fine. **BUT** the only way to learn how to code is by coding yourself



You **NEED** to understand coding to be able to complete M3

<https://github.com/dragos-gruia/MSc-Neuroscience-Python-Course-Development>

Exercises

The level of complexity is described by the number of stars next to the title - the more stars there are, the more complex is the exercise.

1. The RNA complementary (★)

Transcription is the process in which DNA is converted to RNA. To do this, it is necessary to map each letter in a DNA sequence to its complement: A → U, T → A, G → C and C → G. Define a function that takes as input a DNA sequence and returns the RNA complementary.

2. The prime numbers (★)

Define a function that takes as input a list of numbers and returns only those that are prime and that specifies how many they are. Call the function multiple times and check the output to be sure that it is working as expected.

3. Find the GC-content, the individual nucleotides counts and the ATAT motif (★)

In a DNA string you can have four different nucleotides: A, C, T and G. Knowing the number of each individual nucleotides and the percentage of GC-content (G or C nucleotides) is important. Define a function that takes as input a dictionary, with as key the name of a sequence and as value the DNA sequence and:

Schedule of today

Morning

- Two alternatives:
 - Option 1: Review of primer coding exercises + Q&A
 - Option 2: visualization with seaborn or data analysis exercise

Afternoon

Let's code!!

- 8 exercises that review the material you completed in the primer
- You have 20 minutes to solve each exercise
- Don't worry!! We are here to help



Option 1 or Option 2?

1. Unfinished primer exercises
2. Questions about the lectures or the exercises



1. Completed all lectures
2. Completed all primer exercises
3. No questions about the lectures nor the exercises



Time to code!

- 8 exercises
- 3 levels of complexity
- 30 minutes per exercise
- Work alone or in pairs (the more you code alone, the more you learn)

We are here to help!

Exercise 1: the leap years

Write a function that prints the next 20 leap years starting from 2022.

Input: (2022, 20)

Desired output

2024, 2028, 2032, 2036, 2040, 2044, 2048

Exercise 1: Solution



```
def leap_year(year_start, nyears):  
    years = []  
    count_year = 0  
    count_leap_year = 0  
    while count_leap_year < nyears:  
        year = year_start + count_year  
        if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):  
            years.append(year)  
            count_leap_year = count_leap_year+1  
        count_year = count_year+1  
  
    return years  
  
leap_year(2022, 20)
```


Exercise 2: characters numbers

Write a function that calculates the number of characters in each song written by Taylor Swift. Save these in a different series and print them. Also print the highest number of characters that a song had.

Input: ("Data/spotify_taylorswift.csv", "name")


Example songs = pd.Series(['Clean', 'All Too Well', 'Lover', 'Ivy'])

Desired output

5, 12, 5, 3

12

Exercise 2: Solution



```
## CODE HERE
import pandas as pd
import numpy as np

def character_songs(path_to_file, col_title):
    df = pd.read_csv(path_to_file)
    songs_title = df[col_title]
    lengths_songs = []
    for song in songs_title:
        lengths_songs.append(len(song))
    print(pd.Series(lengths_songs))
    print(np.max(lengths_songs))

character_songs("Data/spotify_taylorswift.csv", "name")
```

Exercise 3: popularity more than average

Write a function that goes through each one of Taylor's songs and prints the names of all the ones which have higher than average popularity.

Input: ("Data/spotify_taylorswift.csv", "popularity", "name")

Example songs = pd.Series(['Clean', 'All Too Well', 'Lover', 'Ivy'])

Example popularity = pd.Series([0.2, 0.6, 0.8, 0.4])

Desired output:

All Too Well, Lover

Exercise 3: Solution

```

● ● ●

# OPTION 1: WORKING WITH FOR AND IF LOOPS
def more_popular(path_to_file, col_popularity, col_title):
    df = pd.read_csv(path_to_file)
    average = np.mean(df[col_popularity])
    print(average)
    songs_title = df[col_title]
    songs_popularity = df[col_popularity]
    for i, song in enumerate(songs_title):
        if songs_popularity[i] > average:
            print(song)

# OPTION 2: WORKING WITH PANDAS
def more_popular(path_to_file, col_popularity, col_title):
    df = pd.read_csv(path_to_file)
    average = np.mean(df[col_popularity])

    df_popular = df[df[col_popularity] > average][col_title]
    print(df_popular.tolist())

more_popular("Data/spotify_taylorswift.csv", "popularity", "name")
```

Exercise 4: Find the Vowels☆☆

Write a function that prints the words that contain at least 2 vowels from a series.

Input: `ser = pd.Series(['Apple', 'Orange', 'Plan', 'Python', 'Money'])`

Desired output

Apple, Orange, Money

Exercise 4: Solution



```
def findVowels(ser):  
    vowels = ["a", "e", "i", "o", "u"]  
    for ind, word in enumerate(ser):  
        vow_counts = 0  
        for letter in word:  
            if letter.lower() in vowels:  
                vow_counts = vow_counts + 1  
        if vow_counts >= 2:  
            print(ind, " ", word)  
  
words = pd.Series(['Apple', 'Orange', 'Plan', 'Python', 'Money'])  
findVowels(words)
```

Exercise 5: Pig Latin

Write function that translates a text to Pig Latin and back. English is translated to Pig Latin by taking the first letter of every word, moving it to the end of the word and adding 'ay'.

Input: "The quick brown fox"

Desired output

"Hetay uickqay rownbay oxfay"

Exercise 5: Solution



```
def piglatin(sentence):  
    list_words = sentence.split(" ")  
    new_sent = ""  
    for word in list_words:  
        new_word = word[1:]+word[0].lower()+"ay"  
        new_sent = new_sent + new_word + " "  
    return new_sent  
  
example = "The quick brown fox"  
piglatin(example)
```


Exercise 6: top 10 songs ★★

Write a function that finds the top 10 songs with the highest danceability and then prints the one with the highest popularity.


Input: ("Data/spotify_taylorswift.csv", "name", "danceability", "popularity")

Desired output

['I Think He Knows', 'Treacherous - Original Demo Recording' ...]

Most popular song: Paper Rings

Exercise 6: Solution



```
def top10_songs(path_to_file, col_name, col_beats, col_popular):  
    df = pd.read_csv(path_to_file)  
    dfsort = df.sort_values(by = col_beats, ascending = False)  
    top10 = dfsort[0:10]  
    top10_names = top10[col_name].tolist()  
    print(top10_names)  
    most_pop_song = top10[top10[col_popular] == np.max(top10[col_popular])][col_name]  
    print("\n")  
    print("Most popular song:", most_pop_song.item())  
  
top10_songs("Data/spotify_taylorswift.csv", "name", "danceability", "popularity")
```

Exercise 7: what day is it?

Create a function that outputs the day of the week in which each of Taylor Swift's songs were published.

Input: ("Data/spotify_taylorswift.csv", "release_date")

Example dates = pd.Series(['01 Jan 2010', '02-02-2011', '20120303', '2013/04/04', '2014-05-05'])

Desired output

Date: [0, 1, 2, 3, 4, 5...]

Day of week: [friday', 'friday', 'friday', 'friday', 'friday', ...]

```

def day_of_week(path_to_file, col_date):
    df = pd.read_csv(path_to_file)
    dates = df[col_date]
    days_dict = {0: "Sunday", 1: "Monday", 2: "Tuesday", 3: "Wednesday", 4:
    "Thursday", 5: "Friday", 6: "Saturday"}

    dates_num = []
    days_week = []
    for n, d in enumerate(dates):
        leap_year_this = False
        year = (int(d[2:4]) + (int(d[2:4])//4)) % 7
        monthcode = "033614625035"
        month = int(monthcode[int(d[5:7])])

        if int(d[0:4]) > 1900 and int(d[0:4]) < 2000:
            century = 0
        elif int(d[0:4]) > 2000:
            century = 6

        if int(d[0:4]) in leap_year(1990, 30):
            leap_year_this = True

        total = year + month + century + int(d[8:10])
        if leap_year_this:
            total = total - 1
        total = total % 7
        day_week = days_dict[total]
        dates_num.append(n)
        days_week.append(day_week)
    print("Dates:", dates_num)
    print("Day of week", days_week)

day_of_week("Data/spotify_taylorswift.csv", "release_date")

```

Exercise 7: Solution



You need to use the `leap_year` function that you already defined

Exercise 8: Building a pyramid


Write a function that prints a pyramid-like pattern with numbers starting from 1 and increasing by one each time you go down the pyramid (up to 8)

Input: 8

Desired output

```
1
2 2
3 3 3
4 4 4 4
```

Exercise 8: Solution



```
def pyramid(range_pyr):  
    k = range_pyr - 1  
    for i in range(0, range_pyr):  
        for j in range(0, k):  
            print(end=" ")  
        k = k - 1  
        for j in range(0, i+1):  
            print(i, end=" ")  
        print("\r")  
  
pyramid(8)
```