Andrew Phakeovilay 1B

Programmation efficace

TP3

Table des matières

Partie 1 – prise en main du projet

1. Les données

2196	7630	20221005120000	102250	-20	8	130	7.800000	296.850000	287.850000

Figure 1: Capture de la température 5 octobre à midi à la station de mesure de Blagnac

Le numéro correspond à Blagnac, nous avons bien la date 5 octobre, en température 296.85 K, soit 23.7°C.

2. L’application

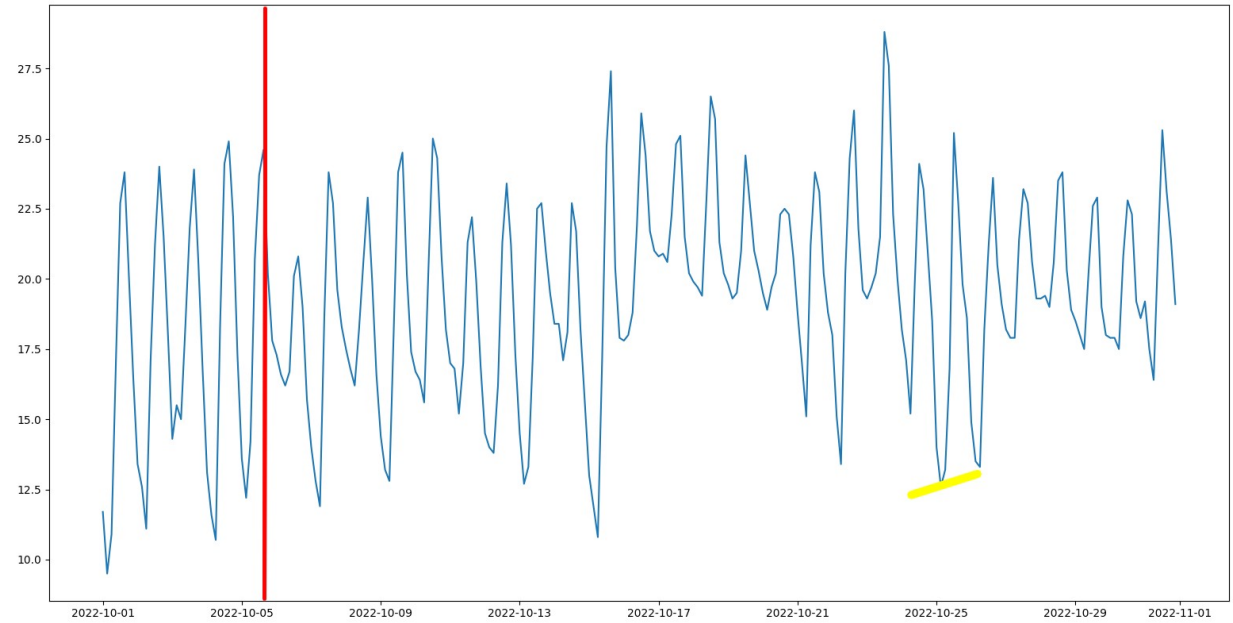

Figure 2: Affichage de l'execution de plot_station, courbe de l'évolution de la température en fonction des mois Oct. - Nov. à Blagnac

3. Mesure du temps écoulé

plot_station.py

```python
def plot_station_temperatures(collection: RecordCollection, station: str):
    print(f'Affichage des temperatures de la station {station}')
    print(f'(1) collecte des donnees')
    x_labels = []
    y_values = []
    debut = datetime.datetime.now()
    for date in collection.dates:
        record = collection.get_record(date, station)
        if record is not None:
            x_labels.append(date)
            y_values.append(record.temperature)
    fin = datetime.datetime.now()
    duree = fin - debut
    print(duree.total_seconds(), 's')
    print(f'(2) affichage de {len(y_values)} temperatures')
    plt.plot(x_labels, y_values)
    plt.show()


def demo(station: str):
    collection = RecordCollection.from_month(10, 2022)
    plot_station_temperatures(collection, station)
```

plot_station_temperatures()

plot_station

```
C:\Users\Andre\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Andre\PycharmProjects\pythonProject\plot_station.py
Affichage des temperatures de la station 07630
(1) collecte des donnees
11.759939 s
(2) affichage de 248 temperatures
```

Figure 3: Capture du code et résultat en secondes

Partie 2 – *Profiling* et amélioration des performances

1. Profiler

2. Analyser

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
3716325	22.28	5.995e-06	38.78	1.044e-05	_strptime.py:309(_strptime)
1	17.59	17.59	17.69	17.69	~:0(<method 'mainloop' of '_tkinter.tkapp' objects>)
3716325	2.929	7.882e-07	41.71	1.122e-05	_strptime.py:565(_strptime_datetime)
3716327	2.882	7.756e-07	2.882	7.756e-07	~:0(<built-in method _locale.setlocale>)
3716327	2.207	5.939e-07	2.924	7.869e-07	locale.py:396(normalize)
3731250	1.828	4.9e-07	1.828	4.9e-07	data.py:31(__getitem__)
3718070	1.75	4.708e-07	1.75	4.708e-07	~:0(<method 'match' of 're.Pattern' objects>)
3716354	1.607	4.325e-07	1.607	4.325e-07	~:0(<method 'groupdict' of 're.Match' objects>)
3716325	1.523	4.099e-07	43.24	1.163e-05	~:0(<built-in method strptime>)
3716327	1.504	4.047e-07	8.506	2.289e-06	locale.py:587(getlocale)

Figure 5: 10 lignes représentant où le programme prend du temps

On peut voir qu'il a 440 milles appels à la fonction _strptime présente dans la classe RecordCollection dans le fichier data.py. On a 2 fonctions dans la classe Record. L'appel à eux appelle date qui appelle strptime. Il faut donc réduire l'appel à date.

```
    def get_record(self, date: datetime, station: str) -> Optional[Record]:
        for r in self:
            if r.date == date and r.station == station:
                return r
        return None

    def get_records_by_date(self, date: datetime) -> List[Record]:
        result = []
        for record in self:
            if record.date == date:
                result.append(record)
        return result
```

Figure 6: méthodes qui posent problèmes

3. Corriger

On va utiliser un dictionnaire pour lier les éléments et accéder plus rapidement.

On crée un nouvel attribut dans le __init__ qui sera initialisé par la méthode qu'on va créer.

```
    def __init__(self, records: List[Record]):
        self.records = records
        self.dicDate = self.constructDicDate()
```

Figure 7: Ajout du nouvel attribut

On crée une fonction constructDicDate. On attribut la date clé, une liste des records lié à la date.

```
    def constructDicDate(self):
        dicContructed = {}
        for attribute in self:
            if attribute.date in dicContructed.keys():
                dicContructed[attribute.date].append(attribute)
            else:
                dicContructed[attribute.date] = [attribute]
        return dicContructed
```

Figure 8: Nouvelle fonction constructDicDate

On change ensuite les fonctions qui ralentissaient l'exécution du programme.

```
    def get_record(self, date: datetime, station: str) -> Optional[Record]:
        for r in self.dicDate[date]:
            if r.station == station:
                return r
        return None

    def get_records_by_date(self, date: datetime) -> List[Record]:
        return self.dicDate[date]
```

Figure 9: Modification des fonctions existantes

On obtient à l'exécution

```
Affichage des temperatures de la station 07630
(1) collecte des donnees
0:00:00.098022 s
(2) affichage de 248 temperatures
```

Figure 10: Nouveau résultat, on passe de 11 s à 0.098 s

Partie 3 – Téléchargement et cache de fichier

```python
@classmethod
def from_month(cls, month: int, year: int) -> 'RecordCollection':
    file = file_pattern.format(year, month)
    if not os.path.exists(file):
        url = 'https://donneespubliques.meteofrance.fr/donnees_libres/Txt/Synop/Archive/' + file.split('/')[1]
        print('Téléchargement', url)
        urllib.request.urlretrieve(url, file)
        if not os.path.exists(file):
            raise Exception('Problème survenu')
    return cls.from_file(file)
```

Figure 11: Modification de la fontion from_month