# Gestion des dates

Laurent Signac – CC-BY-SA – 06-09-21 0844 f402a33e3e44dd2c415d

Manipuler et gérer des dates est une activité courante en analyse de données. Dès qu'on lit des fichiers de mesures, par exemple, il est probable qu'il y ait dedans quelquechose qui s'apparente à des dates. ou des horaires.

Python possède un type de données capable de représenter les horodatages : `datetime` et les intervalles de temps : `timedelta`. À noter qu'il existe aussi un type `date` (juste pour les dates) et un type `time`, juste pour les horaires, et un moyen de gérer les fuseaux horaires, mais nous n'en parlerons pas plus ici (et les outils de la bibliothèque standard ne font pas l'unanimité).

Quel que soit le format utilisé pour les dates dans des fichiers, nous prenons ici le parti de **transformer les données en `datetime` Python le plus tôt possible**. C'est à dire qu'on fera le moins possible de manipulations de dates dans un format autre que le format natif Python.

Dans ces conditions, nous allons détailler ici :

- les opérations courantes utilisables avec `datetime`
- la transformation d'objets divers (chaînes ou timestamps) en `datetime` et inversement

## 1 Importation du module

Le module de gestion des dates est `datetime`. Il contient 3 types de données (entre autres) : `date`, `datetime`, et `timedelta`. Le type `datetime` a le même nom que le module… ce qui peu prêter à confusion. On suppose dans toute la suite qu'on a fait un :

```
>>> from datetime import date, datetime, timedelta
```

## 2 Créer directement un objet datetime

```
>>> datetime(2022, 3, 14) # Le 14 mars 2022 à 0h00
>>> datetime(2022, 3, 14, 15, 10) # Le 14 mars 2022 à 15h10
>>> datetime(2022, 3, 14, 15, 10, 30) # Le 14 mars 2022 à 15h10 et 30 secondes
```

## 3 Transformation depuis et vers `datetime`

### 3.1 Depuis une chaîne

La fonction `datetime.strptime` permet de *parser* une chaîne de caractères ayant un format arbitraire.

```
>>> stri = "15:10-14/03/2022" # pi day en 2022
>>> piday = datetime.strptime(stri, "%H:%M-%d/%m/%Y")
# datetime.datetime(2022, 3, 14, 15, 10)
```

La liste des codes permettant de décrire le format de la date est disponible dans la documentation en ligne : https://docs.python.org/fr/3/library/datetime.html#strftime-strptime-behavior

Voici les plus courants :

| code | signification |
|---|---|
| `%d` | numéro du jour dans le mois sur 2 chiffres |
| `%m` | numéro du mois sur 2 chiffres |
| `%y` | numéro de l'année sur 2 chiffres |
| `%Y` | numéro du l'année sur 4 chiffres |
| `%H` | heures (/24) sur 2 chiffres |

| code | signification |
|---|---|
| `%M` | minutes sur 2 chiffres |
| `%S` | secondes sur 2 chiffres |
| `%w` | jour de la semaine en chiffre |

### 3.2 Vers une chaîne

Inversement, si on doit produire un fichier de données, on peut transformer l'objet `datetime` en chaîne avec la méthode `strftime`, qui utilise le même système de codes que `strptime` :

```
>>> pidaylunch = datetime(2022, 3, 14, 12, 30)
>>> pidaylunch.strftime("le %A %d/%m de l'an %Y à %H:%M:%S")

"le lundi 14/03 de l'an 2022 à 12:30:00"
```

Si le réglage de la langue sur le système est incorrect, le jour de la semaine pourrait s'afficher dans une autre langue.

### 3.3 Depuis un timestamp (*Epoch* Unix)

Un timestamp courant en informatique est la mesure du nombre de secondes écoulées depuis l'*Epoch* (1er janvier 1970 00h00). Comme la plupart des langages, Python possède des fonction dédiées.

```
>>> freedom_ts = 1656608400.0
>>> datetime.fromtimestamp(freedom_ts)
datetime.datetime(2022, 6, 30, 19, 0)
```

### 3.4 Vers un timestamp

On peut bien sûr faire l'inverse :

```
>>> pidaylunch = datetime(2022, 3, 14, 12, 30)
>>> pidaylunch.timestamp()
1647257400.0
```

### 3.5 Format ISO 8601

Il y a une représentation normalisée des dates : ISO 8601. Elle est assez souvent utilisée, et quitte à produire des chaînes de caractères représentant des dates, il est probablement avisé d'utiliser ce format.

```
>>> pidaylunch = datetime(2022, 3, 14, 12, 30)
>>> pidaylunch.isoformat()
'2022-03-14T12:30:00'
```

À noter que le format ISO 8601 peut contenir des informations de fuseau horaire :

```
>>> isodate = '2022-03-14T11:30:00+01:00'
>>> datetime.fromisoformat(isodate)
datetime.datetime(2022, 3, 14, 11, 30, tzinfo=datetime.timezone(datetime.timedelta(seconds=3600
```

## 4 Manipulation des dates

Une fois qu'on a des objets de type `datetime` en Python, on peut obtenir des informations dessus ou faire de l'arithmétique sur les dates. Voici quelques exemples :

```
>>> piday = datetime(2022, 3, 14)
>>> freedom_day = datetime(2022, 6, 30)
>>> piday.day # récupération du jour (idem month, year, hour...)
14
>>> piday.weekday() # jour de la semaine
0  # c'est un lundi
>>> freedom_day.weekday()
3 # c'est un jeudi
>>> laps = freedom_day - piday # différence entre 2 dates donne un timedelta
# datetime.timedelta(days=108)
>>> laps.total_seconds()
9331200.0 # un peu plus de 9 millions de secondes entre les 2 dates
```

Il y a naturellement beaucoup d'autres outils disponibles. On trouvera une liste ici : https://docs.python.org/fr/3/library/datetime.html