Manipulation de dates

Le module **datetime** de Python est essentiel pour manipuler les dates, les heures et les durées. Il regroupe plusieurs classes permettant de représenter, comparer, formater ou calculer des valeurs temporelles avec une grande précision. Il est souvent utilisé dans les applications où la gestion du temps est importante, comme les journaux d'événements, les planifications ou les statistiques temporelles.

```
from datetime import date, time, datetime, timedelta
import pytz
```

La classe **date** représente une date composée de l'année, du mois et du jour. Par exemple, `date(2024, 1, 1)` crée un objet correspondant au 1er janvier 2024. La méthode `date.today()` renvoie la date du jour selon le système local. Ces objets permettent aussi des calculs : on peut leur ajouter ou soustraire un intervalle de temps pour obtenir une autre date.

```
# Créer une instance de date représentant le 1er janvier 2024
date_2024 = date(2024, 1, 1)
print(f"Date créée : {date_2024}")

# Récupérer la date d'aujourd'hui et l'afficher
date_aujourdhui = date.today()
print(f"Date d'aujourd'hui : {date_aujourdhui}")
```

La classe **time** représente uniquement une heure (heures, minutes, secondes et microsecondes), sans notion de jour. Ainsi, `time(14, 30, 45)` correspond à 14h30 et 45 secondes. Pour obtenir l'heure actuelle, on utilise `datetime.now().time()`. Il est possible d'accéder séparément aux attributs comme `hour`, `minute` ou `second`.

```
# Créer une instance de time représentant 14h30 et 45 secondes
temps = time(14, 30, 45)
print(f"Temps créé : {temps}")
```

La classe **datetime** combine à la fois une date et une heure. C'est l'objet le plus complet du module car il permet de manipuler un instant précis. Par exemple, `datetime(2024, 5, 17, 15, 30)` représente le 17 mai 2024 à 15h30. Cet objet peut être converti en texte grâce à la méthode `strftime`, ou reconstruit à partir d'une chaîne grâce à `strptime`. Dans l'exemple, `"2024-10-14"` est converti en objet date avec le format `"%Y-%m-%d"`, qui suit la norme **ISO 8601** — un standard international garantissant la cohérence d'écriture des dates et heures dans le format `YYYY-MM-DD` pour les dates et `HH:MM:SS` pour les heures.

```
heure_actuelle = datetime.now().time()
print(f"Heure actuelle : {heure_actuelle.hour}h {heure_actuelle.minute}m
{heure_actuelle.second}s")
```

```
# Convertir la chaîne "2024-10-14" en un objet date
date_chaine = datetime.strptime("2024-10-14", "%Y-%m-%d").date()
print(f"Date convertie : {date_chaine}")

# Convertir la chaîne "31-12-2024" (format DD-MM-YYYY)
date_chaine_ddmm = datetime.strptime("31-12-2024", "%d-%m-%Y").date()
print(f"Date convertie (format DD-MM-YYYY) : {date_chaine_ddmm}")
```

La classe **timedelta** représente une durée, c'est-à-dire la différence entre deux dates ou heures. Par exemple, `timedelta(days=45)` désigne un intervalle de 45 jours. On peut ainsi calculer une date future : `date.today() + timedelta(days=45)` donnera la date dans 45 jours. Le même principe s'applique pour calculer un nombre de jours entre deux dates en soustrayant deux objets `datetime`, comme dans la différence entre le 1er et le 10 mars 2020 à Montréal.

```
# Calculer la date 45 jours après aujourd'hui
delta_45_jours = date.today() + timedelta(days=45)
print(f"Date après 45 jours : {delta_45_jours}")

# Créer deux objets datetime pour le 1er mars 2020 et le 10 mars 2020 à
Montréal
montreal_tz = pytz.timezone('America/Montreal')

date_debut = datetime(2020, 3, 1, 15, 0, tzinfo=montreal_tz)
date_fin = datetime(2020, 3, 10, 15, 0, tzinfo=montreal_tz)

difference = date_fin - date_debut
print(f"Nombre de jours entre les deux dates : {difference.days}")
```

La classe **tzinfo** gère les fuseaux horaires. Python ne contient pas en standard la base complète des zones mondiales, mais le module externe `pytz` permet d'utiliser des zones telles que `'Europe/Paris'` ou `'Asia/Tokyo'`. En créant des objets `datetime` avec ces zones, on peut afficher simultanément l'heure actuelle à Paris et à Tokyo, ce qui est crucial pour les applications internationales.

```
# Créer un objet datetime pour l'heure actuelle à Paris et Tokyo
paris_tz = pytz.timezone('Europe/Paris')
tokyo_tz = pytz.timezone('Asia/Tokyo')

heure_actuelle_paris = datetime.now(paris_tz)
heure_actuelle_tokyo = datetime.now(tokyo_tz)

print(f"Heure actuelle à Paris : {heure_actuelle_paris}")
print(f"Heure actuelle à Tokyo : {heure_actuelle_tokyo}")
```

Enfin, la bibliothèque **dateutil**, et plus particulièrement son objet `relativedelta`, complète le module en permettant des calculs plus complexes, comme l'ajout de mois ou d'années entières. Par exemple, ajouter

deux mois à la date du 1er février 2024 donne le 1er avril 2024, calcul qu'un simple `timedelta` ne pourrait pas faire puisqu'il ne connaît pas la longueur des mois.

```
# Créer une date représentant le 1er février 2024, puis ajouter 2 mois
date_fevrier = datetime(2024, 2, 1)
date_ajoutee = date_fevrier + relativedelta(months=2)
print(f"Date après ajout de 2 mois : {date_ajoutee}")
```

Ainsi, le module `datetime` forme une base solide pour tout travail temporel en Python. Il permet d'exprimer, convertir et comparer les dates et heures avec précision, tout en respectant les conventions internationales pour garantir la cohérence et l'interopérabilité des données temporelles.