

Ces exercices visent à développer de l'aisance dans l'écriture de code **PYTHON**. Certains sont adaptés à la spécialité mathématiques, quand d'autres en excèdent le niveau d'exigence : la mention « NSI » les identifiera. La mention « T^{ale} » pourra également venir préciser le niveau principalement concerné.

Des exemples de résultats seront souvent fournis pour vous aider à évaluer votre code.

« BONUS PERMANENT » EN NSI :

- ▶ toujours déterminer les préconditions que doivent vérifier les paramètres de vos fonctions;
- ▶ dériver des exemples de résultats (lorsqu'ils sont fournis) une ou plusieurs **assertions**¹. **Attention :** il est souvent **impossible de tester l'égalité de quantités numériques**² !

INFORMATIONS

Dans ce document :

- ▶ une heure est représentée par une liste PYTHON de trois éléments : des nombres entiers, correspondant respectivement et successivement aux heures, minutes et secondes.
Exemple : [23, 12, 57] désigne 23 h 12 min 57 s;
- ▶ de même, une date est représentée par une liste PYTHON comportant trois éléments : des nombres entiers, représentant respectivement et successivement l'année, le mois et le jour.
Exemple : [2021, 03, 01] correspond au 1^{er} mars 2021.

Vous utiliserez les valeurs ci-dessous pour tester vos programmes : les assertions proposées dans les exercices y feront également appel afin de vous aider à valider le bon comportement de vos codes.

HEURE1 = [1, 2, 3]	Remarque : 2000 et 2020 sont	DATE6 = [2021, 6, 30]
HEURE2 = [1, 2, 59]	bissextils; 2021 et 2100 non !	DATE7 = [2021, 7, 31]
HEURE3 = [1, 59, 3]	DATE1 = [2000, 1, 31]	DATE8 = [2021, 4, 31]
HEURE4 = [23, 2, 3]	DATE2 = [2020, 2, 28]	DATE9 = [2021, 9, 30]
HEURE5 = [1, 59, 59]	DATE3 = [2100, 3, 31]	DATE10 = [2021, 10, 31]
HEURE6 = [23, 59, 59]	DATE4 = [2021, 4, 30]	DATE11 = [2021, 11, 30]
HEURE7 = [0, 0, 0]	DATE5 = [2021, 5, 31]	DATE12 = [2021, 12, 31]

EXERCICE 1



Écrire une fonction `heure2str()` ayant comme paramètre une heure, passée sous forme de liste PYTHON (cf. *supra*) et renvoyant une chaîne de caractères convenablement formatée représentant l'heure.

AIDE : on utilisera la fonction `str()` de PYTHON pour réaliser la conversion en chaîne de caractères. Rappelons qu'on accole les chaînes de caractères en les additionnant avec l'opération + (on parle de *concaténation*).

EXEMPLE : `heure2str(HEURE6)` renvoie la chaîne de caractères "23 h 59 min 59 s".

BONUS NSI : gérer l'affichage AM/PM, avec un comportement par défaut qui donne l'affichage sur 24 heures.

EXERCICE 2



Écrire une fonction `date2str()` ayant comme paramètre une date, passée sous forme de liste PYTHON (cf. *supra*) et renvoyant une chaîne de caractères convenablement formatée représentant la date.

AIDE : la fonction `str()` de PYTHON convertit un « objet PYTHON » en chaîne de caractères. On accole les chaînes de caractères en les « additionnant » avec l'opération + (on parle de *concaténation*).

EXEMPLE : `date2str(DATE1)` renvoie la chaîne de caractères "31 janvier 2000".

BONUS : faire en sorte que `heure2str([2021, 1, 1])` donne "1er janvier 2021".

1. Une assertion permet de vérifier la conformité d'une situation : si la vérification échoue, une erreur survient (on dit qu'une *exception est levée*). Par exemple, il est prudent de s'assurer qu'une quantité est non-nulle avant de calculer son inverse...

2. Lorsque la situation implique des calculs numériques pouvant amener des résultats décimaux (comme par exemple un calcul de moyenne, de variance ou d'écart-type), **il faut chercher à proposer des assertions adaptées** : en pratique, on vérifie que la valeur absolue de l'écart entre le résultat théorique et le résultat approché est faible. Cela donnerait une instruction telle que `assert abs(ma_fonction(parametre) - valeur_theorique) < 10**-6`, "Oups", à adapter selon les circonstances.

EXERCICE 3



Écrire une fonction `annee_est_bissextile()` testant si une date appartient à une année bissextile : elle renverra donc le *booléen* `True` ou `False` selon la situation. Une année est bissextile (c'est-à-dire qu'elle compte 366 jours, *via* l'ajout du 29 février) :

- ▶ si l'année est divisible par 4 et non divisible par 100;
- ▶ **ou** si l'année est divisible par 400.

Rappel : « divisible » signifie que la division donne un résultat entier (le reste de la division est donc nul).

EXEMPLE : `annee_est_bissextile(2000)` et `annee_est_bissextile(2004)` donnent `True`. *A contrario*, `annee_est_bissextile(2003)` et `annee_est_bissextile(2100)` renvoient `False`.

EXERCICE 4



Écrire une fonction `ajouter_annees()` ayant comme paramètres une date (donnée sous forme de liste PYTHON, cf. *supra*), suivie d'un entier *relatif* représentant un nombre d'années à ajouter (si ce nombre est négatif, cela reviendra à retrancher du temps). La fonction renverra une liste représentant une *nouvelle* date. On supposera qu'on n'a pas le problème des historiens, et que l'année zéro est permise!

EXEMPLE : `ajouter_annees(2000, -2000)` renverra `[0, 1, 31]`.

EXERCICE 5



Écrire une fonction `ajouter_heures()` ayant comme paramètres une heure (donnée sous forme de liste PYTHON, cf. *supra*), suivie d'un entier *relatif* représentant un nombre d'heures à ajouter (ou à *retrancher* si le nombre est négatif). La fonction renverra une liste représentant une *nouvelle* heure.

AIDE : l'opérateur % donne le reste d'une division entière. Ainsi, $-12 \% 8$ égale 4 car $-12 = -2 \times 8 + 4$ (noter que tout se passe bien avec des entiers négatifs également).

EXEMPLE : `ajoute_heures(HEURE6, 1)` donne `[0, 59, 59]`; `ajoute_heures(HEURE1, 24)` renvoie `[1, 2, 3]` et `ajoute_heures(HEURE1, -2)` a pour résultat `[23, 2, 3]`.

EXERCICE 6



Écrire une fonction `ajouter_minutes()` ayant comme paramètres une heure, suivie d'un entier *relatif* représentant un nombre de minutes à ajouter (ou à *retrancher* si ce nombre est négatif). La fonction renverra une liste représentant une *nouvelle* heure.

AIDE : l'opérateur // donne le quotient *entier* d'une division *entière*. Ainsi, $-12 // 8$ a pour résultat -2 car $-12 = -2 \times 8 + 4$ (noter que tout se passe bien avec des entiers négatifs également).

EXEMPLE : `ajoute_minutes(HEURE1, 58)` renvoie `[2, 0, 3]` quand `ajoute_minutes(HEURE1, -63)` donne `[23, 59, 3]`.

EXERCICE 7



Écrire une fonction `ajouter_secondes()` ayant comme paramètres une heure (une liste PYTHON, cf. *supra*), suivie d'un entier *relatif* représentant un nombre de secondes à ajouter (si ce nombre est négatif, cela reviendra à *retrancher* du temps). La fonction renverra une liste représentant une *nouvelle* heure.

EXEMPLE : `ajoute_secondes(HEURE6, 1)` donne `[0, 0, 0]`, `ajoute_secondes(HEURE7, -1)` renvoie `[23, 59, 59]`. Enfin, `ajoute_secondes(HEURE1, 86400)` renvoie `[1, 2, 3]` (1 jour = 86 400 secondes).

EXERCICE 8



Écrire une fonction `ajouter_mois()` ayant comme paramètres une date, suivie d'un entier *relatif* représentant un nombre de mois à ajouter (si ce nombre est négatif, cela reviendra à retrancher du temps). La fonction renverra une liste représentant une *nouvelle* date. On supposera encore qu'on n'a pas le problème des historiens, et que l'[année zéro](#) est permise!

AIDE : si un calcul nous amène au mois de février, on « réduira » (si nécessaire) le numéro du jour conformément au nombre de jours que possède le mois de février cette année-là.

EXEMPLE : `ajouter_mois(DATE1, -1)` renvoie `[1999, 12, 31]` alors que, l'année 2000 étant bissextile (février comptant alors 29 jours), `ajouter_mois(DATE1, 1)` renverra `[2000, 2, 29]`. Autres exemples : `ajouter_mois(DATE1, 13)` donne `[2001, 2, 28]`. On peut également « remonter très loin le temps » : `ajouter_mois(DATE1, -24001)` renvoie `[-1, 12, 31]`.

EXERCICE 9



Écrire une fonction `ajouter_jours()` ayant comme paramètres une date suivie d'un entier *relatif* représentant un nombre de jours à ajouter (si ce nombre est négatif, cela reviendra à *retrancher* du temps). La fonction renverra une liste représentant une *nouvelle* date. On négligera le problème des historiens (l'[année zéro](#) est permise).

EXEMPLE : `ajouter_jours(DATE1, -2000)` renverra `[0, 1, 31]`.

EXERCICE 10



Écrire une fonction `ajouter_date()` ayant comme paramètres **deux** dates et renvoyant une liste PYTHON représentant une *nouvelle* date. On négligera le problème des historiens (l'[année zéro](#) est permise).

AIDE : on pourra, si on le souhaite, utiliser les fonctions définies dans les exercices précédents.

EXEMPLE : `ajouter_date(DATE1, HEURE1)` renvoie `[2001, 4, 3]`³.

BONUS NSI : écrire une fonction `ajouter_dates()` qui permet d'ajouter plusieurs dates. On va ici **bien au-delà** du programme : [ce lien](#) et [celui-ci](#) vous seront utiles pour comprendre le « dépaquetage de structures ».

EXERCICE 11



Temps UNIX

Écrire une fonction `date2unix()` ayant comme paramètres une date. La fonction renverra un entier représentant le [temps UNIX](#) (le nombre de secondes écoulées depuis le 1^{er} janvier 1970 à 0 h 0 min 0 s [UTC](#), hors [secondes intercalaires](#)). On ne tiendra pas compte des fuseaux horaires, et on considérera toujours qu'il sera 0 h 0 min 0 s pour le jour concerné!

EXEMPLE : `date2unix(DATE1)` renverra 949276800.

EXERCICE 12



Temps UNIX réciproque

Écrire une fonction `unix2date()` faisant l'exact contraire de la fonction précédente.

EXEMPLE : `unix2date(date2unix(DATE1))` renverra `[2000, 1, 31]` (soit la valeur désignée par `DATE1`).

3. Cela fonctionnera car l'ordinateur n'a ici aucun moyen de « savoir » qu'une liste de 3 valeurs représente une heure et non une date! On gère ce problème en NSI, avec des dictionnaires en 1^{re}, avec la *Programmation Orientée Objets* en terminale.