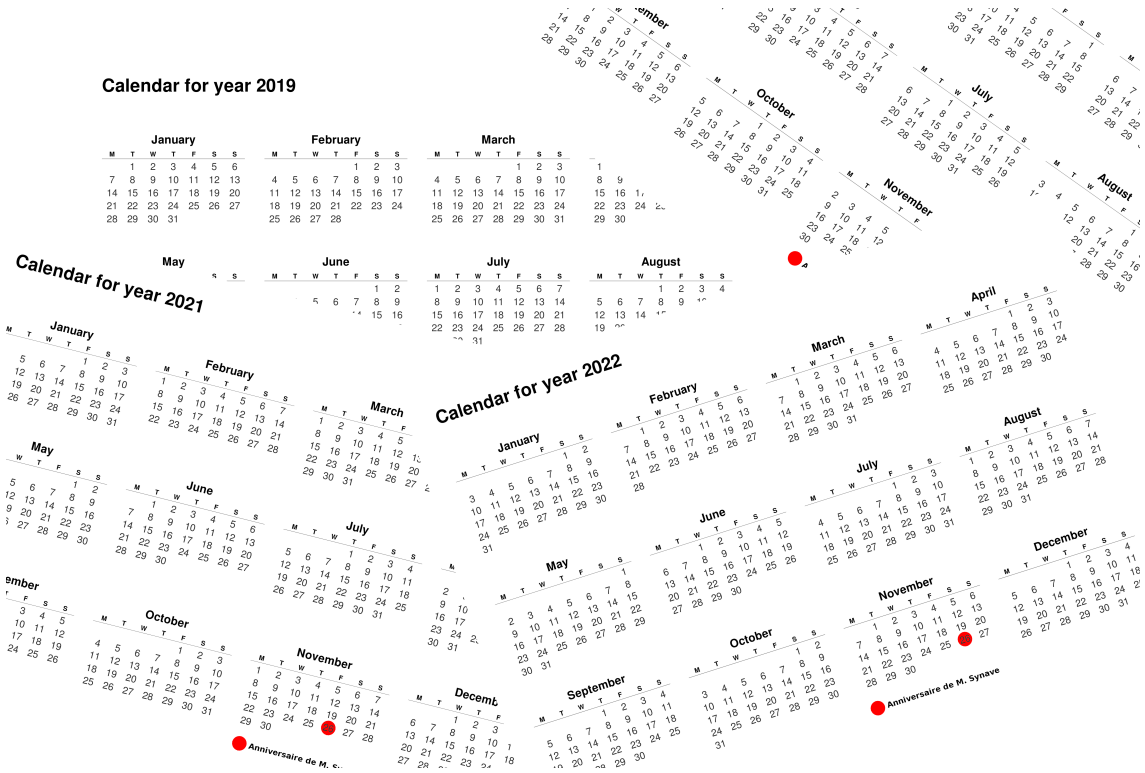
SAÉ 1.01 - Implémentation d'un besoin client

Gestion d'un calendrier perpétuel

Comme ça, tu ne seras plus jamais en retard en cours!



Descriptif détaillé de la SAÉ

En quoi consiste cette SAÉ ?

En partant d'un besoin exprimé par un client, l'objectif est de réaliser une application qui réponde à ce besoin. Cette SAÉ permet une première mise en pratique du développement autour d'un besoin client.

Quelles sont les productions de cette SAÉ ?

— Code de l'application.
— Traces d'exécution des jeux d'essais.

Quelles sont les compétences développées ?

Développer des applications informatiques simples :
— AC 1 Implémenter des conceptions simples.
— AC 3 Faire des essais et évaluer leurs résultats en regard des spécifications.

Projet de 12h, travail **individuel**.

Objectifs

L'objet de ce TP est de vous faire réaliser un calendrier. Toutes les entrées-sorties du programme seront en mode "console" et une partie du travail est facultative.

Ce sujet est découpé en deux parties, la première est obligatoire, elle ne comporte pas de difficulté particulière et vous serez guidé pour sa réalisation alors que la deuxième partie contient des parties facultatives dont l'implémentation est laissée libre.

Barème : les deux parties seront notées sur 10. Réaliser uniquement la partie obligatoire permet d'obtenir donc juste la moyenne, à condition qu'elle fonctionne et que le code soit bien commenté. Pour obtenir une note supérieure à 10, il faut avoir réalisé une portion de la partie facultative. La note maximale peut être atteinte même si l'ensemble de la partie facultative n'est pas réalisée (par exemple toutes les questions "faciles", plusieurs "moyennes" et une question "difficile").

⚠ **Veuillez lire tout le sujet avant de commencer à développer !**

Partie obligatoire

1 Coder des dates

Remarque : il existe des modules de python qui permettent de représenter des dates et des calendriers (par exemple `datetime`). Pour ce travail, il vous est demandé de ne pas les utiliser.

Vous allez coder votre propre module qui permet de coder des dates. Vous pourrez ensuite utiliser ce module dans vos programmes. Par exemple :

```
from date import *

annee = int(input ('entrez une annee : '))
print(annee , ' est bissextile : ', est_bissextile(annee))
```

Le module date aura la structure suivante :

```
def est_bissextile(annee: int) -> bool :
    ...

def valide_simple(d: Tuple[int,int,int]) -> bool :
    ...
```

Afin de vous guider dans la réalisation du module `date`, sa structure vous est donnée (fichier `date.py`). Pour chaque fonction, vous pouvez consulter sa documentation ainsi que son utilisation. La syntaxe de `doctest` est utilisée. Par exemple pour la fonction `est_bissextile`, on retrouve :

```
def est_bissextile(annee: int) -> bool :
    '''''''
    retourne vrai si l'année est bissextile
    >>> est_bissextile(2020)
    True
    >>> est_bissextile(2021)
    False
    >>> est_bissextile(2022)
    False
    >>> est_bissextile(1900)
    False
    >>> est_bissextile(2000)
    True
    '''''''
```

Ceci indique que les années 2000 et 2020 doivent être considérées comme bissextiles mais

pas 1900, 2021 et 2022. Votre travail consistera à écrire le code pour que ces tests fonctionnent. Vous pouvez exécuter tous les tests du module en exécutant simplement le programme contenu dans le fichier `date.py` grâce à Visual Studio Code.

Vous obtiendrez un rapport de l'exécution des tests sur votre module. Il est normal, au début que l'ensemble des test échouent, cela va s'arranger petit à petit au fur et à mesure du développement du module.

Exemple :

```
**********************************************************************
4 items had failures:
   2 of   5 in __main__.compare
   3 of   5 in __main__.est_bissextile
   4 of  12 in __main__.valide_complet
   3 of   8 in __main__.valide_simple
30 tests in 5 items.
18 passed and 12 failed.
***Test Failed*** 12 failures.
```

1.1 Année bissextile

Ecrivez le code de la fonction `est_bissextile` qui retourne vrai si l'année est bissextile et faux sinon.

```
def est_bissextile(annee: int) -> bool :
```

1.2 Création d'une date

Ecrivez le code de la fonction `cree_date` qui retourne une variable de type dictionnaire représentant la date passée en paramètre sous forme de trois entiers. Vous utiliserez les noms que vous souhaitez pour les entrées. Si vous êtes en manque d'inspiration, utilisez simplement `jour`, `mois` et `annee`.

De plus, vous vérifierez que les paramètres sont bien des entiers. Dans le cas contraire, la fonction retournera `None`.

```
def cree_date(j: int, m: int, a: int) -> Dict :
```

1.3 Copie d'une date

Ecrivez le code de la fonction `copie_date` qui retourne une variable de type dictionnaire représentant une copie de la date passée en paramètre. **Cette fonction devra être utilisée lorsque vous voudrez copier une date pour, par exemple, faire des calculs sur la date sans modifier la variable de départ.**

```
def copie_date(date: Dict) -> Dict :
```

1.4 Comparaison de deux dates

Ecrivez la fonction `compare` qui va comparer deux dates, date1 et date2, passées en paramètre.

Le retour attendu est le suivant : $\begin{cases} -1 & \text{si la date1 est avant la date2} \\ +1 & \text{si la date1 est après la date2} \\ 0 & \text{si le date1 et date2 sont égales} \end{cases}$

```
def compare(d1: Dict, d2: Dict) -> int :
```

1.5 Vérifier la validité des dates

La fonction `valide_simple` permet de vérifier que le dictionnaire passé en paramètre correspond à une date valide. Pour commencer, on vérifie simplement que :

— Le jour est un entier compris entre 1 et 31
— Le mois est un entier compris entre 1 et 12

```
def valide_simple(d: Dict) -> bool :
```

Ensuite, écrivez la fonction `valide_complet` qui :

1. appelle la fonction `valide_simple`,
2. si le résultat est valide, vérifie que le nombre de jour du mois ne dépasse pas le maximum (tenir compte des mois de 30 jours et des années bissextiles pour février).

```
def valide_complet(d: Dict) -> bool :
```

2 Codage des calendriers

Un calendrier sera représenté comme une liste de dictionnaires. Pour chaque élément de la liste, pour chaque dictionnaire, une entrée sera la `date` et l'autre l'`évènement`.

Ajoutez maintenant une procédure :

```
def ajoute_calendrier(calendrier: List, date: Dict, description: str ) -> NoReturn :
```

Attention : avant d'ajouter la date au calendrier, vous devez vérifier que c'est une date valide.

Ajoutez enfin une procédure qui permet d'afficher le calendrier :

```
def affiche_calendrier(calendrier: List) -> NoReturn :
```

Vous pouvez maintenant créer le fichier `prog.py` et le programme principal ci-dessous. Faites le nécessaire pour pouvoir l'exécuter.

```
# définition du dictionnaire
calendrier = []
# ajout des évènements
ajoute_calendrier(calendrier, cree_date(1,1,2022), ''Jour de l'an'')
ajoute_calendrier(calendrier, cree_date(1,5,2022), ''Fête du travail'')
ajoute_calendrier(calendrier, cree_date(8,5,2022), ''Armistice 1945'')
ajoute_calendrier(calendrier, cree_date(14,7,2022), ''Fête nationale'')
ajoute_calendrier(calendrier, cree_date(15,8,2022), ''Assomption'')
ajoute_calendrier(calendrier, cree_date(25,12,2022), ''Noel'')
# affichage
affiche_calendrier(calendrier)
```

Le résultat attendu :

```
Le 1/1/2022 : Jour de l'an
Le 1/5/2022 : Fête du travail
Le 8/5/2022 : Armistice 1945
Le 14/7/2022 : Fête nationale
Le 25/8/2022 : Assomption
Le 25/12/2022 : Noel
```

Partie facultative

Pour chaque partie, nous proposons une liste non exhautive des ajouts ou modifications possibles plus ou moins classées par ordre de difficulté. Il n'est absolument pas nécessaire de les implémenter tous : réalisez ce qui vous semble réalisable, et surtout, **testez toutes les fonctionnalités que vous ajoutez avec la syntaxe doctest.**

Vous pouvez ajouter toutes les fonctions que vous jugerez nécessaires mais le fonctionnement de la partie précédente doit rester identique.

3 Ajouts possibles aux dates

3.1 Facile - I'm too young to die

Ajoutez une procédure qui permet de changer le format d'affichage. Par exemple :

```
def format_affichage(affichage: str) -> NoReturn :
```

avec `affichage`, une chaine de caractères : "fr" ou "gb".
Si on choisit "fr", l'affichage sera sous la forme `jj/mm/aaaa` alors que, si on choisit "gb", l'affichage sera `mm/jj/aaaa`.

3.2 Moyen - Hurt me plenty

Ajoutez une fonction qui calcule le jour de la semaine d'une date (lundi, mardi ...).

```
def calcule_jour(date: Dict) -> str :
```

3.3 Moyen - Hurt me plenty

Ajoutez une fonction qui calcule la date du lendemain et de la veille.

```
def calcule_veille_lendemain(date: Dict) -> Tuple[Dict, Dict] :
```

3.4 Difficile - Ultra-Violence

Ajoutez une fonction qui ajoute ou retranche n jours à la date.

```
def ajoute_n_jour(date: Dict, n: int) -> Dict :
```

```
def retranche_n_jour(date: Dict, n: int) -> Dict :
```

3.5 Difficile - Ultra-Violence

Ajoutez une fonction qui calcule le nombre de jours écoulés entre deux dates passées en paramètre.

```
def ecart_jour(date: Dict, date: Dict) -> int :
```

4 Ajouts possibles au calendrier

4.1 Facile - I'm too young to die

Ajoutez une procédure qui permet d'ajouter automatiquement toutes les fêtes fixes (dont la date ne change pas comme Noel et Nouvel an) d'une année donnée. L'année sera passée en paramètre de la fonction.

```
def ajoute_fetes(calendrier: List, annee: int) -> NoReturn :
```

4.2 Moyen - Hurt me plenty

Modifiez la procédure `ajoute_calendrier` pour que le calendrier soit trié en permanence. Pour cela, utilisez la méthode `compare` pour savoir à quelle moment insérer la date dans la liste.

4.3 Moyen - Hurt me plenty

Ajoutez une fonction `trouve_evenement` qui recherche le nom d'un évenement dont on connait la date. La méthode `trouve_evenement` permet d'avoir le choix dans la date et de retourner l'évenement qui correspond. Si aucun événement n'est trouvé alors la fonctione retournera `None`.

```
def trouve_evenement(calendrier: List, date: Dict) -> str :
```

4.4 Très difficile - Nightmare!

Ajoutez une procédure qui permet d'ajouter automatiquement toutes les fêtes mobiles (pâques, la pentecôte, etc.) d'une année donnée. L'année sera passée en paramètre de la méthode.

```
def ajoute_fetes_mobiles(calendrier: List, annee: int) -> NoReturn :
```

5 Autres modifications possibles

5.1 Moyen - Hurt me plenty

Ajoutez une procédure `afficheur_mensuel` prenant un calendrier, un mois et une année en paramètre. L'affichage se fera sous forme d'une liste des jours :

```
** mai 2021 **
  samedi 01 : Fête du travail
dimanche 02 :
   lundi 03 :
   mardi 04 :
mercredi 05 :
   jeudi 06 :
vendredi 07 :
  samedi 08 : Armistice 1945
dimanche 09 :
      ...
```

5.2 Difficile - Ultra-Violence

Ajoutez une procédure `afficheur_mensuel_tableau` prenant un calendrier et un mois en paramètre. L'affichage se fera sous forme d'un tableau :

```
-------------------------------------------------------------
             ** mai 2021 **                | sam 01 | dim 02
                                           |fete tra|
-------------------------------------------------------------
lun 03 | mar 04 | mer 05 | jeu 06 | ven 07 | sam 08 | dim 09
       |        |        |        |        |arm 1945|
-------------------------------------------------------------
lun 10 | mar 11 | mer 12 | jeu 13 | ven 14 | sam 15 | dim 16
       |        |        |Ascensio|        |        |
-------------------------------------------------------------
lun 17 | mar 18 | mer 19 | jeu 20 | ven 21 | sam 22 | dim 23
       |        |        |        |        |        |Pentecot
-------------------------------------------------------------
lun 24 | mar 25 | mer 26 | jeu 27 | ven 28 | sam 29 | dim 30
 férié |        |        |        |        |        |
-------------------------------------------------------------
lun 31 |
       |
-------
```

5.3 Facile - I'm too young to die

Quand ont été utilisés ces niveaux de difficulté pour la première fois ? Ajoutez une procédure permettant d'ajouter l'événement relatif à la sortie du premier jeu dans lequel ces niveaux de difficulté ont été utilisés à un calendrier passé en paramètre.

6 À rendre

Une seule archive .zip (ou tar.gz) qui porte votre nom dans laquelle il y aura :

— Le fichier `date.py` que vous aurez complété.

— Un fichier **texte** README.txt (ou .md) qui comportera vos nom et prénom et décrira les modifications optionnelles que vous avez réalisées (en reprenant les codes de la partie facultative). Vous pouvez aussi préciser toute modification complémentaire que vous avez jugée utile d'apporter, des difficultés que vous avez rencontrées dans le projet ou des choix que vous avez faits.

Tout projet qui ne respecterait pas ces contraintes, qui ne pourrait pas être exécuté ou qui ne sera pas commenté suffisamment ne sera pas corrigé.

7 Références pour les algorithmes

7.1 Trouver le jour de la semaine d'une date

Plusieurs méthodes sont proposées, choisissez la plus simple !

`https://fr.wikibooks.org/wiki/Curiosit%C3%A9s_math%C3%A9matiques/Trouver_le_jour_de_la_semaine_avec_une_date_donn%C3%A9e`

7.2 Trouver la date de Pâques

`https://fr.wikipedia.org/wiki/Calcul_de_la_date_de_P%C3%A2ques`

8 Credits

Sujet très fortement inspiré d'un sujet créé par Samuel Delepoulle aka Manwald#8515