CORRIGÉ SUJET N°3

- CYCLE MENSTRUEL -

Le Dossier ZIP de l'épreuve pratique N°3 se situe à l'adresse suivante :
https://sujets.examens-concours.gouv.fr/delos/api/file/public/69c4f0247dc1be7ea38d75a4

Ce corrigé est tiré du site suivant :
https://clem2429.github.io/Correction-Sujets-NSI-2026/

La version du fichier Python corrigé est accessible à l'adresse suivante :
https://clem2429.github.io/Correction-Sujets-NSI-2026/corrige/cycle_menstruel_corrige.py

- Ce sujet est le N°3 de la banque des sujets pratiques de NSI 2026;
- Ce sujet (corrigé) comporte 4 questions ;
- Chaque réponse à une question se trouve sur une page distincte.

QUESTION 1

Écrire en Python une fonction nommée `est_bissextile` qui prend en paramètre un entier correspondant à une année et qui renvoie un booléen indiquant si elle est bissextile en appliquant la règle donnée ci-dessus.

Pour rappel, et en simplifiant l'énoncé :
- Une année divisible par 400 est bissextile ;
- Une année divisible par 4 mais pas par 100 est bissextile,
- Les autres ne le sont pas.

```
def est_bissextile(annee):
    '''Renvoie un booléen indiquant si l'année en argument est
bissextile.'''
    if annee%400 == 0:
        return True
    elif annee%4 == 0 and annee%100 != 0:
        return True
    else:
        return False

# Version plus courte :
def est_bissextile(annee):
    '''Renvoie un booléen indiquant si l'année en argument est
bissextile.'''
    return annee%400 == 0 or (annee%4 == 0 and annee%100 != 0)
```

QUESTION 2

Écrire en Python une fonction nommée `determiner_phase` qui prend en paramètre un entier(compris entre 1 et 28 inclus) qui correspond au jour d'un cycle et qui renvoie un entier correspondant au numéro de la phase associée. À l'aide d'une assertion, on garantira que l'entier donné en argument est compris entre 1 et 28 inclus.

Pour rappel :
- J 1 à 5 : phase 1 ;
- J 6 à 13 : phase 2 ;
- J 14 : phase 3 ;
- J 15 à 28 : phase 4.

```
def determiner_phase(i):
    '''Renvoie le numéro de la phase associée au jour i d'un
cycle.'''
    assert 1 <= i <= 28, "Le jour doit être compris entre 1 et
28 inclus."
    if 1 <= i <= 5:
        return 1
    elif 6 <= i <= 13:
        return 2
    elif i == 14:
        return 3
    else:
        return 4
```

QUESTION 3

La fonction `ajouter_jours`, dont le code est déjà fourni, prend en paramètres une date et un entier représentant un nombre de jours. Elle renvoie la nouvelle date obtenue après ajout de ces jours.
Compléter la fonction test_ajouter_jours en ajoutant au moins trois autres tests pertinents. Pour chaque test ajouté, une brève justification doit être donnée afin d'expliquer pourquoi ce cas est important à vérifier.
Dans le but d'ajouter l'ensemble des dates de début de règles sur un agenda en ligne pour une année donnée, on souhaite créer un fichier au format iCalendar. La structure d'un tel fichier pour un calendrier est la suivante :

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID: *le nom du calendrier*
*une suite d'événements*
END :VCALENDAR
```

Chaque événement d'une journée est lui-même décrit par une entrée de la forme suivante :

```
BEGIN :VEVENT
DTSTART: la date JJ/MM/AAAA écrite sous la forme AAAAMMJJ
SUMMARY : la description de l'événement
END :VEVENT
```

Ci-dessous, vous retrouverez 12 tests différents (dont le 1er est tiré du sujet donné). Ces tests devraient tester la plupart des situations pouvant exister cependant, seuls trois sont demandés !

```
def test_ajouter_jours():
    # 1 - (Déjà présent) Ajout de jours dans le même mois :
    assert ajouter_jours((7, 9, 2025), 3) == (10, 9, 2025),
"Echec Test 1"

    #2 - Ajout de jours qui font changer de mois (30 jours):
    assert ajouter_jours((25, 4, 2025), 10) == (5, 5, 2025),
"Echec Test 2"

    #3 - Ajout de jours qui font changer de mois (31 jours):
    assert ajouter_jours((25, 7, 2025), 10) == (4, 8, 2025),
"Echec Test 3"

    #4 - Ajout de jours qui font changer d'année (365 jours):
    assert ajouter_jours((25, 12, 2025), 10) == (4, 1, 2026),
"Echec Test 4"

    #5 - Ajout de jours qui font changer d'année (366 jours) :
    assert ajouter_jours((25, 12, 2024), 10) == (4, 1, 2025),
"Echec Test 5"

    #6 - Ajout de jours qui font changer de mois et d'année (40
jours) :
    assert ajouter_jours((25, 12, 2024), 40) == (3, 2, 2025),
"Echec Test 6"

    #7 - Ajout de jours qui font changer de mois et d'année
(vers février en année bissextile) :
    assert ajouter_jours((25, 12, 2024), 66) == (1, 3, 2025),
"Echec Test 7"

    #8 - Ajout d'une année complète (année bissextile) :
    assert ajouter_jours((1, 1, 2024), 366) == (1, 1, 2025),
"Echec Test 8"

    #9 - Ajout d'une année complète (année non bissextile) :
    assert ajouter_jours((1, 1, 2025), 365) == (1, 1, 2026),
"Echec Test 9"

    #10 - Ajout de jours en février dans une année bissextile :
    assert ajouter_jours((25, 2, 2024), 5) == (1, 3, 2024),
"Echec Test 10"
```

```
    #11 - Ajout de jours en février dans une année non
bissextile :
    assert ajouter_jours((25, 2, 2025), 5) ==  (2, 3, 2025),
"Echec Test 11"

    #12 - Ajout de 0 jour :
    assert ajouter_jours((25, 2, 2024), 0) == (25, 2, 2024),
"Echec Test 12"
```

QUESTION 4

On complète les nombres strictement inférieurs à 10 par un 0 pour s'assurer que la valeur de DTSTART soit toujours de longueur 8. Ainsi, la date du 3 juillet 2026 s'écrit sous la forme `DTSTART : 20260703`.

La fonction `calendrier_``cycles``, présente dans le fichier fourni, prend en paramètre une date correspondant au premier jour des dernières règles et renvoie la liste chronologique des dates de début de règles qui se présentent dans les 100 jours suivant cette date, date incluse, au format iCalendar sous la forme d'une chaîne de caractères.
Observer avec la fonction test_calendrier_cycles que le calendrier renvoyé par la fonction calendrier_cycles n'est pas dans un format valide.
Identifier le problème dans la fonction calendrier_cycles, proposer une démarche de résolution et la mettre en œuvre.

On remarque que le problème, quand on exécute `test_calendrier_cycles()`, se trouve au niveau de l'enregistrement de la date :

```
date = str(annee)+str(mois)+str(jour)
```

En effet, si le jour ou le mois est inférieur à 10, alors la date ne sera pas au format AAAAMMJJ. Par exemple, pour le 3 juillet 2026, on obtiendra `202673` au lieu de `20260703`.

OPTION 1

```
# Fonction corrigée :
def calendrier_cycles(date_regles):
    """Renvoie une chaîne de caractère contenant au format
iCalendar, l'ensemble
    des dates de début de règles qui se présentent dans les 100
jours suivants
    `date_regles`, date incluse.

    Hypothèse : cycle régulier de 28 jours. """

    cal_lignes = ['BEGIN:VCALENDAR', 'VERSION:2.0', 'PRODID:']

    date_courante = date_regles
    jours_ecoules = 0

    # On ajoute les dates tant que l'on ne dépasse pas 100
jours écoulés
    while jours_ecoules <= 100:
        jour, mois, annee = date_courante
        cal_lignes.append('BEGIN:VEVENT')
        cal_lignes.append('SUMMARY: Règles')
        # CORRECTION :
        date = str(annee)+str(mois).zfill(2)+str(jour).zfill(2)
        cal_lignes.append('DTSTART:'+date)
        cal_lignes.append('END:VEVENT')
        date_courante = ajouter_jours(date_courante, 28)
        jours_ecoules += 28

    cal_lignes.append('END:VCALENDAR')

    # La méthode join va renvoyer ici une unique chaîne
contenant toutes les
    # chaînes de la liste séparées par des sauts de lignes.
    return '\n'.join(cal_lignes)
```

OPTION 2

```
# Fonction corrigée :
def calendrier_cycles(date_regles):
    """Renvoie une chaîne de caractère contenant au format
iCalendar, l'ensemble
    des dates de début de règles qui se présentent dans les 100
jours suivants
    `date_regles`, date incluse.

    Hypothèse : cycle régulier de 28 jours. """

    cal_lignes = ['BEGIN:VCALENDAR', 'VERSION:2.0', 'PRODID:']

    date_courante = date_regles
    jours_ecoules = 0

    # On ajoute les dates tant que l'on ne dépasse pas 100
jours écoulés
    while jours_ecoules <= 100:
        jour, mois, annee = date_courante
        cal_lignes.append('BEGIN:VEVENT')
        cal_lignes.append('SUMMARY: Règles')
        # CORRECTION :
        date = ""
        str_mois = ""
        str_jour = ""
        if 0 < len(str(annee)) < 4:
            return "Année invalide, elle doit contenir 4
chiffres."
        if len(str(mois)) < 2:
            str_mois = "0"+str(mois)
        else:
            str_mois = str(mois)
        if len(str(jour)) < 2:
            str_jour = "0"+str(jour)
        else:
            str_jour = str(jour)
        date = str(annee) + str_mois + str_jour
        # FIN DE CORRECTION
        cal_lignes.append('DTSTART:'+date)
        cal_lignes.append('END:VEVENT')
        date_courante = ajouter_jours(date_courante, 28)
        jours_ecoules += 28
```

```
    cal_lignes.append('END:VCALENDAR')

    # La méthode join va renvoyer ici une unique chaîne
contenant toutes les
    # chaînes de la liste séparées par des sauts de lignes.
    return '\n'.join(cal_lignes)
```

ATTENTION :
Par ailleurs, on remarquera que la fonction est en fait encore plus faussée : elle n'affiche pas jusqu'à 100 jours mais avant du fait du :
`while jours_ecoules + 28 <= 100:`
Que l'on corrige en :
`while jours_ecoules <= 100:`

On mettait en fait fin à la période trop tôt, à savoir après 84 jours au lieu de 112 jours. En corrigeant ce problème, on obtient 4 dates de début de règles au lieu de 3.