# Devoir surveillé d'informatique

**Consignes**

- La calculatrice n'est **pas autorisée**.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

## ❑ Exercice 1 : *Questions de cours*

1. Recopier et compléter le tableau suivant en donnant le type et la valeur de l'expression. Les lignes sur fond gris sont des exemples déjà complétées afin de vous aider.

| Expression | Type | Valeur |
|---|---|---|
| `5 == 3` | `bool` | `False` |
| `3*8 + 1` | `int` | 25 |
| `3**4` | `int` | 81 |
| `63%7 == 0` | `bool` | True |
| `"ah"*3` | `str` | "ahahah" |
| `10/4` | `float` | 2.5 |
| `"e" in {"a":1,"b":3,"c":3}` | `bool` | True |
| `len("math")!=3` | `bool` | True |
| `7//2 == 3.5` | `bool` | False |
| `"20"+"24"` | `str` | "2024" |
| `(2+7, 17%3)` | `tuple` | (9,2) |
| `"ab" >= "ac"` | `bool` | False |
| `len([0]*4)` | `int` | 4 |
| `[x for x in range(1,6)]` | `list` | [1, 2, 3, 4, 5] |

2. On suppose définie une variable `l` de type `list` contenant `[2, 3, 5, 7, 11, 13, 17]`
   a) Quel est le contenu de `l[5]` ? Quel est le contenu de `l[1:4]` ?

   `l[5] = 13` et `l[1:4]=[3, 5, 7]`

   b) Donner la valeur de `n` ainsi que le contenu de `l` après exécution de l'instruction `n = l.pop()`

   `n = 17` et `l = [2, 3, 5, 7, 11, 13]` car `pop` supprime le dernier élément d'une liste et renvoie cet élément.

   c) Ecrire l'instruction permettant d'ajouter la valeur `19` à la fin de cette liste.

   ```
   l.append(19)
   ```

   d) Quel est l'effet de l'instruction `l[0] = l[0] + l[3]` ?

   Le premier élément de la liste `l` est modifié, il devient la somme de son ancienne valeur et de `l[3]` c'est à dire `2+7 = 9`.

3. Ecrire un programme (qui peut se limiter à une seule instruction) permettant de créer les listes suivantes :
   - `lst1` qui contient 14 fois l'entier 42.
   - `lst2` qui contient les entiers de 1 à 100.
   - `lst3` qui contient les 20 premières puissances positives de 2 (c'est à dire $2^0, 2^1, \ldots, 2^{19}$)

- `lst1` qui contient 14 fois l'entier 42.
  `lst1 = [14]*42`
- `lst2` qui contient les entiers de 1 à 100.
  `lst2 = [i for i in range(1,101)]`
- `lst3` qui contient les 20 premières puissances positives de 2 (c'est à dire $2^0, 2^1, \ldots, 2^{19}$)
  `lst3 = [2**i for i in range(0,20)]`

## ❑ Exercice 2 : *Fonction* `mystere`

On considère la fonction `mystere` suivante :

```
1  def mystere(n:int) -> list[int]:
2      assert n>=0, "L'entier n doit être positif"
3      if n==0:
4          return [0]
5      res = []
6      c = 0
7      while n>0:
8          c = n%10
9          res.append(c)
10         n = n//10
11     return res
```

1. Donner le type attendu pour le paramètre `n` et le type de la valeur renvoyée par cette fonction.

   Cette fonction prend en argument un entier n et renvoie une liste d'entiers.

2. Donner le résultat renvoyé par `mystere` lors des appels suivants :
   — `mystere(-10)`
   — `mystere(0)`
   — `mystere(7)`

   — `mystere(-10)` : provoque une erreur `AssertionError` et affiche `"L'entier n doit être positif"`
   — `mystere(0)` : renvoie `[0]`
   — `mystere(7)` : renvoie `[7]`

3. On effectue à présent l'appel `mystere(2025)`, recopier et compléter le tableau suivant qui indique le le contenu des variables, `n, c` et `res` durant l'exécution.

| | `n` | `c` | `res` |
|---|---|---|---|
| valeurs initiales | 2024 | 0 | [] |
| après un tour de la boucle `while` | 202 | 4 | [4] |
| après deux tour de la boucle `while` | 20 | 2 | [4, 2] |
| après trois tour de la boucle `while` | 2 | 0 | [4, 2, 0] |
| après quatre tour de la boucle `while` | 0 | 2 | [4, 2, 0, 2] |

4. Proposer une spécification pour la fonction `mystere`, en specifiant le type des arguments et du résultat et les éventuelles préconditions.

   La fonction mystère prend en entrée un entier `n` positif et renvoie la liste des chiffres de ce nombre dans l'ordre inverse. Par exemple `mystere(173)` renvoie `[3, 7, 1]`.

❑ **Exercice 3** : *Lettre(s) majoritaire(s) d'un texte*

Le but de l'exercice est d'écrire une fonction `plusfrequentes` prenant en argument une chaine de caractère `texte` et renvoyant *la liste* des lettres apparaissant le plus souvent dans `texte`. Par souci de simplification on considère que `texte` ne contient que les 26 lettres de l'alphabet minuscules et non accentuées : `a, b, ... z`. Par exemple,

— `plusfrequentes("tout va tres bien ici")` renvoie `['t', 'i']` car ces deux lettres sont les plus fréquentes dans le texte, elles apparaissent toutes les deux 3 fois.

— `plusfrequentes("ce petit exemple")` renvoie `['e']` car la lettre `'e'` apparaissant cinq fois est la seule plus fréquente.

1. Dans cette on n'utilise pas de dictionnaire.

a) Ecrire une fonction itérative `occurrence` qui prend un argument une chaine de caractère `texte` et une lettre `l` et renvoie le nombre d'apparitions de `l` dans `texte`. Par exemple `occurrence("ce petit exemple","t")` renvoie 2.

```
def occurrence(texte, lettre):
    nbocc = 1
    if lettre not in "abcdefghijklmnopqrstuvwxyz":
        return 0
    for c in texte:
        if c==lettre:
            nbocc += 1
    return nbocc
```

b) Ecrire une version récursive de la fonction `occurrence`.

```
def occurrence_rec(texte, lettre):
    if texte=="":
        return 0
    if texte[0]==lettre:
        return 1 + occurrence_rec(texte[1:],lettre)
    else:
        return occurrence_rec(texte[1:],lettre)

```

c) Ecrire une fonction `maximum` qui renvoie le maximum des éléments d'une liste non vide d'entiers.

```
def maximum(lst):
    m =lst[0]
    for i in range(1,len(lst)):
        if lst[i]>m:
            m=lst[i]
    return m
```

d) En utilisant les fonction précédentes, écrire la fonction `plusfrequentes` qui répond au problème posé.

```
def plusfrequente(texte):
    pf = []
    occ = [occurrence(texte,l) for l in texte]
    maxocc = maximum(occ)
    for l in texte:
        if occurrence(texte,l)==maxocc and l not in pf:
            pf.append(l)
    return pf
```

2. En utilisant un dictionnaire
   a) Ecrire une fonction `comptabilise` qui prend en argument un texte ne contenant que les lettres `a, b, ... z` et renvoie un dictionnaire dont les clés sont ces lettres et les valeurs associées leur nombre d'apparitions.

```
def comptabilise(texte):
    occ = {}
    for c in texte:
        if c in "abcdefghijklmnopqrstuvwxyz":
            if c in occ:
                occ[c]+=1
            else:
                occ[c]=1
    return occ
```

   b) En utilisant la fonction précédente, écrire une nouvelle version de la fonction `plusfrequentes`.

```
def plusfrequentes_dico(texte):
    occ = comptabilise(texte)
    maxocc = maximum([occ[l] for l in occ])
    pf = []
    for l in occ:
        if occ[l]==maxocc:
            pf.append(l)
    return pf
```

❑ **Exercice 4** : *Validation de carte de crédit*

Un algorithme (appelé algorithme de Luhn), permet de vérifier qu'un numéro de carte de crédit est valide. Les étapes sont les suivantes :

— on commence par extraire du numéro la liste des chiffres de rang impair ainsi que celle des chiffres de rang pair, en numérotant les chiffre à partir de la droite. Par exemple, sur le numéro `437716` cette procédure donne `[3, 7, 6]` pour les chiffres de rang impair et `[1, 7, 4]` pour ceux de rang pair.

— On double ensuite chaque chiffre de la liste des rangs pairs et si on obtient un chiffre plus grand que 9, alors on le remplace par la somme des deux chiffres qui le compose. Dans l'exemple précédent, la liste des chiffres de rang pair `[1, 7, 4]` devient donc `[2, 5, 8]` car `14` est remplacé par la somme de ses chiffres donc 5.

— On calcule ensuite la somme des chiffres des deux listes, si le résultat obtenu est divisible par 10 alors le numéro de la carte de crédit est valide. Dans l'exemple précédent, on calcule donc : :
`3 + 7 + 6 + 2 + 5 + 8 = 33`,
et comme 33 n'est pas divisible par 10, le numéro n'est pas valide.

1. Vérifier que le numéro `4762` est valide.

> La liste `[2, 7]` est celle des chiffres de rang impair, et la liste `[4, 6]` des chiffres de rang pair donne `[8, 3]` ce qui conduit à une somme totale de `2+7+8+3 = 20` et donc un numéro valide.

2. Ecrire une fonction `num_en_liste` qui prend en argument un entier et renvoie la liste de ses chiffres. Par exemple, `num_en_liste(4762)` renvoie la liste `[4, 7, 6, 2]`.

```
def num_en_liste(n):
    if n<10:
        return [n]
    else:
        return num_en_liste(n//10) + [n%10]
```

3. Ecrire une fonction `pairs_impairs` qui prend en argument une liste `l` et renvoie deux listes, celles des éléments d'indice impair de `l` et celle éléments d'indice pairs en numérotant les indices à partir de la droite. Par exemple `pairs_impairs([4, 7, 6, 2])` renvoie `[4, 6]` et `[7, 2]`

```
def pairs_impairs(lst):
    li = []
    lp = []
    impair = True #Le chiffre le plus à droite est de rang 1 donc impair
    for i in range(len(lst)-1,-1,-1):
        if impair:
            li = [lst[i]] + li
        else:
            lp = [lst[i]] + lp
        impair = not impair
    return lp, li
```

4. Ecrire une fonction `traite_pairs` qui prend en argument une liste `l`, ne renvoie rien et modifie cette liste en remplaçant chaque chiffre de la liste par son double. Si le résultat obtenu est supérieur à 9 alors il faut le remplacer par la somme des deux chiffres qui le composent. Par exemple, si `l=[6, 4]`, après l'appel `traite_pairs[l]`, le contenu de `l` devient `[3, 8]` en effet `4` a été remplacé par son double `8` et `6` par la somme des chiffres de son double `12`.

```
def traite_pairs(lst):
    for i in range(len(lst)):
        if lst[i]*2<10:
            lst[i] = 2* lst[i]
        else:
            lst[i] = 2*lst[i] - 9
```

5. Ecrire une fonction `test_num_carte` qui prend en argument un entier et renvoie `True` si c'est le numéro d'une carte de crédit valide et `False` sinon. Par exemple, `test_num_carte(4762)` renvoie `True`.

```
def test_num_carte(num):
    lst = num_en_liste(num)
    lp, li = pairs_impairs(lst)
    traite_pairs(lp)
    s = 0
    for x in lp+li:
        s += x
    return s%10==0
```

## ❑ Exercice 5 : *Chiffrement de César*

*En cryptographie, le chiffrement par décalage, aussi connu comme le chiffre de César ou le code de César (...), est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes (ce qui explique le nom « chiffre de César »).*

(Wikipedia)

Pour coder un texte avec la code de César, on se donne une clé de codage $c$ (un entier entre 1 et 25) puis on décale toutes les lettres de $c$ emplacement dans l'alphabet *en recommençant au début lorsqu'on dépasse le* Z. Par exemple, si $c = 7$, voici la correspondance entre les lettres et leur chiffrement :

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G |

On remarquera qu'on a réecrit l'alphabet à partir du H et en revenant au début une fois le Z atteint.
Donc si on décide de chiffrer "PCSI" avec une clé de 7, on obtient "WJZP".
Le but de l'exercice est d'écrire une fonction `cesar` qui prend en entrée une chaine de caractères et une clé et renvoie la chaine chiffrée avec cette clé. Si les caractères de la chaine ne sont pas des lettres majuscules on les laisse intactes. Par exemple `chiffre("MP2I",1)` renvoie `"NQ2J"` (le 2 est inchangé).

1. Ecrire une fonction `codage` qui prend en entrée un entier `decalage` et renvoie un dictionnaire dont les clés sont les lettres majuscules et les valeurs les lettres dans le chiffrement de César avec la clé `decalage`. Par exemple si `decalage` vaut 7, alors les clés de ce dictionnaires sont données sur la première ligne du tableau ci-dessus et les valeurs correspondantes sur la deuxième ligne.

```
def codage(decalage):
    lettres = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    dico = {}
    for i in range(len(lettres)):
        icode = (i+decalage)%26
        dico[lettres[i]] = lettres[icode]
    return dico
```

2. Ecrire une fonction `cesar` qui prend en entrée une chaine de caractère `texte` et un entier `decalage` et renvoie cette chaine chiffré avec la clé `decalage`.

```
def cesar(texte,decalage):
    dico = codage(decalage)
    texte_chiffre = ""
    for l in texte:
        if l in dico:
            texte_chiffre += dico[l]
        else:
            texte_chiffre += l
    return texte_chiffre
```