# Épreuve pratique 💻

Vous trouverez ci-dessous un série de sujets de l'épreuve pratique, disponibles publiquement sur la Banque Nationale des Sujets (novembre 2021).

Une nouvelle version (qui sera *a priori* en grande partie semblable à celle-ci) sera publiée en janvier 2022 sur le site Eduscol.

## 0.1. Exercice 04.1 ☑

### Exercice 04.1

**Énoncé**

Écrire une fonction qui prend en paramètre un tableau d'entiers non vide et qui renvoie la moyenne de ces entiers. La fonction est spécifiée ci-après et doit passer les assertions fournies.

```
def moyenne (tab):
    '''
    moyenne(list) -> float
    Entrée : un tableau non vide d'entiers
    Sortie : nombre de type float
    Correspondant à la moyenne des valeurs présentes dans le
    tableau
    '''

assert moyenne([1]) == 1
assert moyenne([1,2,3,4,5,6,7] == 4
assert moyenne([1,2]) == 1.5
```

**Correction**

```
1  def moyenne(tab):
2      '''
3      moyenne(list) -> float
4      Entrée : un tableau non vide d'entiers
5      Sortie : nombre de type float
6      Correspondant à la moyenne des valeurs présentes dans le
7      tableau
8      '''
9      somme = 0
10     for elt in tab:
11         somme += elt
12     return somme / len(tab)
```

## 0.2. Exercice 04.2 ☑

## Exercice 04.2

### Énoncé

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.

L'algorithme traite le cas du tableau vide.

L'algorithme est écrit pour que la recherche dichotomique ne se fasse que dans le cas où la valeur est comprise entre les valeurs extrêmes du tableau.

On distingue les trois cas qui renvoient `False` en renvoyant `False,1` , `False,2` et `False,3`.

Compléter l'algorithme de dichotomie donné ci-après.

```
def dichotomie(tab, x):
    """
    tab : tableau trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    # cas du tableau vide
    if ...:
        return False,1
    # cas où x n'est pas compris entre les valeurs extrêmes
    if (x < tab[0]) or ...:
        return False,2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
        if x > tab[m]:
            debut = m + 1
        else:
            fin = ...
    return ...
```

Exemples :

```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],27)
(False, 3)
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],1)
(False, 2)
>>> dichotomie([],28)
(False, 1)
```

**Correction**

```
def dichotomie(tab, x):
    """
    tab : tableau trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    # cas du tableau vide
    if tab = []:
        return False,1
    # cas où x n'est pas compris entre les valeurs extrêmes
    if (x < tab[0]) or (x > tab[-1]):
        return False,2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = (debut + fin) // 2
        if x == tab[m]:
            return True
        if x > tab[m]:
            debut = m + 1
        else:
            fin = m - 1
    return False
```

## 0.3. Exercice 05.1 ☑

### Exercice 05.1

**Énoncé**

On modélise la représentation binaire d'un entier non signé par un tableau d'entiers dont les éléments sont 0 ou 1. Par exemple, le tableau `[1, 0, 1, 0, 0, 1, 1]` représente l'écriture binaire de l'entier dont l'écriture décimale est `2**6 + 2**4 + 2**1 + 2**0 = 83`.

À l'aide d'un parcours séquentiel, écrire la fonction convertir répondant aux spécifications suivantes :

```
def convertir(T):
    """
    T est un tableau d'entiers, dont les éléments sont 0 ou 1 et
    représentant un entier écrit en binaire. Renvoie l'écriture
    décimale de l'entier positif dont la représentation binaire
    est donnée par le tableau T
    """
```

Exemple :

```
>>> convertir([1, 0, 1, 0, 0, 1, 1])
83
>>> convertir([1, 0, 0, 0, 0, 0, 1, 0])
130
```

**Correction**

```
1  def convertir(T):
2      puissance = 0
3      total = 0
4      for i in range(len(T)-1, -1, -1):
5          total += T[i]*(2**puissance)
6          puissance += 1
7      return total
```

## 0.4. Exercice 05.2 ☑

## Exercice 05.2

### Énoncé

La fonction `tri_insertion` suivante prend en argument une liste `L` et trie cette liste en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

```
def tri_insertion(L):
    n = len(L)

    # cas du tableau vide
    if ...:
        return L
    for j in range(1,n):
        e = L[j]
        i = j

    # A l'étape j, le sous-tableau L[0,j-1] est trié
    # et on insère L[j] dans ce sous-tableau en déterminant
    # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].

        while i > 0 and L[i-1] > ...:
            i = ...

    # si i != j, on décale le sous tableau L[i,j-1] d'un cran
    # vers la droite et on place L[j] en position i

        if i != j:
            for k in range(j,i,...):
                L[k] = L[...]
            L[i] = ...
    return L
```

Exemples :

```
>>> tri_insertion([2,5,-1,7,0,28])
[-1, 0, 2, 5, 7, 28]
>>> tri_insertion([10,9,8,7,6,5,4,3,2,1,0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

### Correction

```
def tri_insertion(L):
    n = len(L)

    # cas du tableau vide
    if L == []:
        return L

```

```
    for j in range(1,n):
        e = L[j]
        i = j

    # A l'étape j, le sous-tableau L[0,j-1] est trié
    # et on insère L[j] dans ce sous-tableau en déterminant
    # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].

        while i > 0 and L[i-1] > e:
            i = i - 1

        # si i != j, on décale le sous tableau L[i,j-1] d'un cran
    # vers la droite et on place L[j] en position i

        if i != j:
            for k in range(j,i,-1):
                L[k] = L[k-1]
            L[i] = e
    return L
```

## 0.5. Exercice 06.1 ☑

### Exercice 06.1

**Énoncé**

On s'intéresse au problème du rendu de monnaie. On suppose qu'on dispose d'un nombre infini de billets de 5 euros, de pièces de 2 euros et de pièces de 1 euro. Le but est d'écrire une fonction nommée `rendu` dont le paramètre est un entier positif non nul `somme_a_rendre` et qui retourne une liste de trois entiers `n1`, `n2` et `n3` qui correspondent aux nombres de billets de 5 euros (`n1`) de pièces de 2 euros (`n2`) et de pièces de 1 euro (`n3`) à rendre afin que le total rendu soit égal à `somme_a_rendre`.

On utilisera un algorithme glouton : on commencera par rendre le nombre maximal de billets de 5 euros, puis celui des pièces de 2 euros et enfin celui des pièces de 1 euros.

Exemples :

```
>>> rendu(13)
[2,1,1]
>>> rendu(64)
[12,2,0]
>>> rendu(89)
[17,2,0]
```

**Correction**

```
1  def rendu(somme_a_rendre):
2      pieces = [5, 2, 1]
3      retour = [0, 0, 0]
4      reste_a_rendre = somme_a_rendre
5      for i in range(3):
6          retour[i] = reste_a_rendre // pieces[i]
7          reste_a_rendre = reste_a_rendre % pieces[i]
8      return retour
```

## 0.6. Exercice 06.2 ☑

*à noter une erreur dans la version officielle, sur la méthode `enfile()`*

### Exercice 06.2

**Énoncé**

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée. On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None
```

Compléter la classe `File suivante` où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

```
class File :
    def __init__(self) :
        self.dernier_file = None

    def enfile(self,element) :
        nouveau_maillon = Maillon(...)
        nouveau_maillon.suivant = self.dernier_file
        self.dernier_file = ...

    def est_vide(self) :
        return self.dernier_file == None

    def affiche(self) :
        maillon = self.dernier_file
        while maillon != ... :
            print(maillon.valeur)
            maillon = ...

    def defile(self) :
        if not self.est_vide() :
            if self.dernier_file.suivant == None :
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = ...
            while maillon.suivant.suivant != None :
                maillon = maillon.suivant
            resultat = ...
            maillon.suivant = None
            return resultat
        return None
```

On pourra tester le fonctionnement de la classe en utilisant les commandes suivantes dans la console Python :

```
>>> F = File()
>>> F.est_vide()
True
>>> F.enfile(2)
>>> F.affiche()
2
>>> F.est_vide()
False
>>> F.enfile(5)
>>> F.enfile(7)
>>> F.affiche()
7
5
2
>>> F.defile()
2
>>> F.defile()
5
>>> F.affiche()
7
```

**Correction**

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None

class File :
    def __init__(self) :
        self.dernier_file = None

    def enfile(self,element) :
        nouveau_maillon = Maillon(element)
        nouveau_maillon.suivant =  self.dernier_file
        self.dernier_file = nouveau_maillon

    def est_vide(self) :
        return self.dernier_file == None

    def affiche(self) :
        maillon = self.dernier_file
        while maillon != None :
            print(maillon.valeur)
            maillon = maillon.suivant

    def defile(self) :
        if not self.est_vide() :
            if self.dernier_file.suivant == None :
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = self.dernier_file
            while maillon.suivant.suivant != None :
                maillon = maillon.suivant
            resultat = maillon.suivant.valeur
```

```
34                maillon.suivant = None
35                return resultat
36            return None
```

## 0.7. Exercice 07.1 ☑

**Exercice 07.1**

### Énoncé

On s'intéresse à la suite d'entiers définie par `U1 = 1`, `U2 = 1` et, pour tout entier naturel `n`, par `Un+2 = Un+1 + Un`.

Elle s'appelle la suite de Fibonacci.

Écrire la fonction `fibonacci` qui prend un entier `n > 0` et qui renvoie l'élément d'indice `n` de cette suite.

On utilisera une programmation dynamique (pas de récursivité).

Exemple :

```
>>> fibonacci(1)
1
>>> fibonacci(2)
1
>>> fibonacci(25)
75025
>>> fibonacci(45)
1134903170
```

### Correction

On utilise un dictionnaire pour stocker au fur et à mesure les valeurs.

```
1  def fibonnaci(n):
2      d = {}
3      d[1] = 1
4      d[2] = 1
5      for k in range(3, n+1):
6          d[k] = d[k-1] + d[k-2]
7      return d[n]
```

## 0.8. Exercice 07.2 ☑

### Exercice 07.2

#### Énoncé

Les variables `liste_eleves` et `liste_notes` ayant été préalablement définies et étant de même longueur, la fonction `meilleures_notes` renvoie la note maximale qui a été attribuée, le nombre d'élèves ayant obtenu cette note et la liste des noms de ces élèves.

Compléter le code Python de la fonction `meilleures_notes` ci-dessous.

```
liste_eleves = ['a','b','c','d','e','f','g','h','i','j']
liste_notes = [1, 40, 80, 60, 58, 80, 75, 80, 60, 24]

def meilleures_notes():
    note_maxi = 0
    nb_eleves_note_maxi = ...
    liste_maxi = ...

    for compteur in range(...):
        if liste_notes[compteur] == ...:
            nb_eleves_note_maxi = nb_eleves_note_maxi + 1
            liste_maxi.append(liste_eleves[...])
        if liste_notes[compteur] > note_maxi:
            note_maxi = liste_notes[compteur]
            nb_eleves_note_maxi = ...
            liste_maxi = [...]

    return (note_maxi,nb_eleves_note_maxi,liste_maxi)
```

Une fois complété, le code ci-dessus donne

```
>>> meilleures_notes()
(80, 3, ['c', 'f', 'h'])
```

#### Correction

```
liste_eleves = ['a','b','c','d','e','f','g','h','i','j']
liste_notes = [1, 40, 80, 60, 58, 80, 75, 80, 60, 24]

def meilleures_notes():
    note_maxi = 0
    nb_eleves_note_maxi = 0
    liste_maxi = []

    for compteur in range(len(liste_eleves)):
        if liste_notes[compteur] == note_maxi:
            nb_eleves_note_maxi = nb_eleves_note_maxi + 1
            liste_maxi.append(liste_eleves[compteur])
        if liste_notes[compteur] > note_maxi:
```

```
            note_maxi = liste_notes[compteur]
            nb_eleves_note_maxi = 1
            liste_maxi = [liste_eleves[compteur]]

    return (note_maxi,nb_eleves_note_maxi,liste_maxi)
```

## 0.9. Exercice 08.1 ☑

### Exercice 08.1

**Énoncé**

Écrire une fonction `recherche` qui prend en paramètres `caractere`, un caractère, et `mot`, une chaîne de caractères, et qui renvoie le nombre d'occurrences de `caractere` dans `mot`, c'est-à-dire le nombre de fois où `caractere` apparaît dans `mot`.

Exemples :

```
>>> recherche('e', "sciences")
2
>>> recherche('i',"mississippi")
4
>>> recherche('a',"mississippi")
0
```

**Correction**

```
def recherche(caractere, mot):
    somme = 0
    for lettre in mot:
        if lettre == caractere:
            somme += 1
    return somme
```

## 0.10. Exercice 08.2 ☑

### Exercice 08.2

#### Énoncé

On s'intéresse à un algorithme récursif qui permet de rendre la monnaie à partir d'une liste donnée de valeurs de pièces et de billets - le système monétaire est donné sous forme d'une liste `pieces=[100, 50, 20, 10, 5, 2, 1]` - (on supposera qu'il n'y a pas de limitation quant à leur nombre), on cherche à donner la liste de pièces à rendre pour une somme donnée en argument. Compléter le code Python ci-dessous de la fonction `rendu_glouton` qui implémente cet algorithme et renvoie la liste des pièces à rendre.

```
pieces = [100,50,20,10,5,2,1]

def rendu_glouton(arendre, solution=[], i=0):
    if arendre == 0:
        return ...
    p = pieces[i]
    if p <= ... :
        solution.append(...)
        return rendu_glouton(arendre - p, solution,i)
    else :
        return rendu_glouton(arendre, solution, ...)
```

On devra obtenir :

```
>>>rendu_glouton(68,[],0)
[50, 10, 5, 2, 1]
>>>rendu_glouton(291,[],0)
[100, 100, 50, 20, 20, 1]
```

#### Correction

```
pieces = [100,50,20,10,5,2,1]

def rendu_glouton(arendre, solution=[], i=0):
    if arendre == 0:
        return solution
    p = pieces[i]
    if p <= arendre :
        solution.append(p)
        return rendu_glouton(arendre - p, solution,i)
    else :
        return rendu_glouton(arendre, solution, i+1)
```

## 0.11. Exercice 15.1 ☑

### Exercice 15.1

**Énoncé**

Écrire une fonction `rechercheMinMax` qui prend en paramètre un tableau de nombres non triés `tab`, et qui renvoie la plus petite et la plus grande valeur du tableau sous la forme d'un dictionnaire à deux clés 'min' et 'max'. Les tableaux seront représentés sous forme de liste Python.

Exemples :

```
>>> tableau = [0, 1, 4, 2, -2, 9, 3, 1, 7, 1]
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': -2, 'max': 9}
>>> tableau = []
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': None, 'max': None}
```

**Correction**

```
1  def rechercheMinMax(tab):
2      if tab==[]:
3          return {'min': None, 'max': None}
4      else:
5          d = {}
6          d['min'] = tab[0]
7          d['max'] = tab[0]
8          for val in tab:
9              if val < d['min']:
10                 d['min'] = val
11             if val > d['max']:
12                 d['max'] = val
13         return d
```

## 0.12. Exercice 15.2 ☑

## Exercice 15.2

### Énoncé

On dispose d'un programme permettant de créer un objet de type `PaquetDeCarte`, selon les éléments indiqués dans le code ci-dessous. Compléter ce code aux endroits indiqués par `#A compléter`, puis ajouter des assertions dans l'initialiseur de `Carte`, ainsi que dans la méthode `getCarteAt()`.

```
class Carte:
    """Initialise Couleur (entre 1 à 4), et Valeur (entre 1 à
13)"""
    def __init__(self, c, v):
        self.Couleur = c
        self.Valeur = v

    """Renvoie le nom de la Carte As, 2, ... 10, Valet, Dame,
Roi"""
    def getNom(self):
        if (self.Valeur > 1 and self.Valeur < 11):
            return str(self.Valeur)
        elif self.Valeur == 11:
            return "Valet"
        elif self.Valeur == 12:
            return "Dame"
        elif self.Valeur == 13:
            return "Roi"
        else:
            return "As"

    """Renvoie la couleur de la Carte (parmi pique, coeur, carreau,
trefle"""
    def getCouleur(self):
        return ['pique', 'coeur', 'carreau', 'trefle'][self.Couleur
- 1]

class PaquetDeCarte:
    def __init__(self):
        self.contenu = []

    """Remplit le paquet de cartes"""
    def remplir(self):
        #A compléter

    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        #A compléter
```

Exemple :

```
>>> unPaquet = PaquetDeCarte()
>>> unPaquet.remplir()
```

```
>>> uneCarte = unPaquet.getCarteAt(20)
>>> print(uneCarte.getNom() + " de " + uneCarte.getCouleur())
8 de carreau
```

**Correction**

Attention, le code proposé ne respecte pas les standards de notation :

- il ne faut pas de majuscules sur les noms des attributs
- la docstring se place à l'intérieur de la fonction et non au dessus.

```
class Carte:
    """Initialise Couleur (entre 1 à 4), et Valeur (entre 1 à
13)"""
    def __init__(self, c, v):
        assert c in range(1,5)
        assert v in range(1,14)
        self.Couleur = c
        self.Valeur = v

    """Renvoie le nom de la Carte As, 2, ... 10, Valet, Dame,
Roi"""
    def getNom(self):
        if (self.Valeur > 1 and self.Valeur < 11):
            return str( self.Valeur)
        elif self.Valeur == 11:
            return "Valet"
        elif self.Valeur == 12:
            return "Dame"
        elif self.Valeur == 13:
            return "Roi"
        else:
            return "As"

    """Renvoie la couleur de la Carte (parmi pique, coeur, carreau,
trefle"""
    def getCouleur(self):
        return ['pique', 'coeur', 'carreau', 'trefle']
[self.Couleur]

class PaquetDeCarte:
    def __init__(self):
        self.contenu = []

    """Remplit le paquet de cartes"""
    def remplir(self):
        for nb_coul in range(1,5):
            for val in range(1,14):
                self.contenu.append(Carte(nb_coul, val))

    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        assert pos<52,'erreur position'
        return self.contenu[pos]
```

## 0.13. Exercice 19.1 ☑

**Exercice 19.1**

**Énoncé**

Écrire une fonction `recherche` qui prend en paramètres un tableau `tab` de nombres entiers triés par ordre croissant et un nombre entier `n`, et qui effectue une recherche dichotomique du nombre entier `n` dans le tableau non vide `tab`. Cette fonction doit renvoyer un indice correspondant au nombre cherché s'il est dans le tableau, `-1` sinon.

Exemples :

```
>>> recherche([2, 3, 4, 5, 6], 5)
3
>>> recherche([2, 3, 4, 6, 7], 5)
-1
```

**Correction**

```
1  def recherche(tab, n):
2      ind_debut = 0
3      ind_fin = len(tab) - 1
4      while ind_debut <= ind_fin:
5          ind_milieu = (ind_debut + ind_fin) // 2
6          if tab[ind_milieu] == n:
7              return ind_milieu
8          elif tab[ind_milieu] < n:
9              ind_debut = ind_milieu + 1
10         else:
11             ind_fin = ind_milieu - 1
12     return -1
```

## 0.14. Exercice 19.2 ☑

### Exercice 19.2

**Énoncé**

Le codage de César transforme un message en changeant chaque lettre en la décalant dans l'alphabet. Par exemple, avec un décalage de 3, le A se transforme en D, le B en E, ..., le X en A, le Y en B et le Z en C. Les autres caractères ('!',' ?'…) ne sont pas codés.

La fonction `position_alphabet` ci-dessous prend en paramètre un caractère `lettre` et renvoie la position de `lettre` dans la chaîne de caractères `ALPHABET` s'il s'y trouve et `-1` sinon. La fonction `cesar` prend en paramètre une chaîne de caractères `message` et un nombre entier `decalage` et renvoie le nouveau message codé avec le codage de César utilisant le décalage `decalage`.

```
1  ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
2
3  def position_alphabet(lettre):
4      return ALPHABET.find(lettre)
5
6  def cesar(message, decalage):
7      resultat = ''
8      for ... in message:
9          if lettre in ALPHABET:
10             indice = ( ... ) % 26
11             resultat = resultat + ALPHABET[indice]
12         else:
13             resultat = ...
14     return resultat
```

Compléter la fonction `cesar`.

Exemples :

```
>>> cesar('BONJOUR A TOUS. VIVE LA MATIERE NSI !',4)
'FSRNSYV E XSYW. ZMZI PE QEXMIVI RWM !'
>>> cesar('GTSOTZW F YTZX. ANAJ QF RFYNJWJ SXN !',-5)
'BONJOUR A TOUS. VIVE LA MATIERE NSI !'
```

**Correction**

```
1  ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
2
3  def position_alphabet(lettre):
4      return ALPHABET.find(lettre)
5
```

```
6  def cesar(message, decalage):
7      resultat = ''
8      for lettre in message:
9          if lettre in ALPHABET:
10             indice = (position_alphabet(lettre) + decalage) % 26
11             resultat = resultat + ALPHABET[indice]
12         else:
13             resultat = resultat + lettre
14     return resultat
```

## 0.15. Exercice 20.1 ☑

### Exercice 20.1

#### Énoncé

On a relevé les valeurs moyennes annuelles des températures à Paris pour la période allant de 2013 à 2019. Les résultats ont été récupérés sous la forme de deux listes : l'une pour les températures, l'autre pour les années :

```
t_moy = [14.9, 13.3, 13.1, 12.5, 13.0, 13.6, 13.7]
annees = [2013, 2014, 2015, 2016, 2017, 2018, 2019]
```

Écrire la fonction `mini` qui prend en paramètres le tableau `releve` des relevés et le tableau `date` des dates et qui renvoie la plus petite valeur relevée au cours de la période et l'année correspondante.

Exemple :

```
>>> mini(t_moy, annees)
(12.5, 2016)
```

#### Correction

{{ correction(True, "

```
t_moy = [14.9, 13.3, 13.1, 12.5, 13.0, 13.6, 13.7]
annees = [2013, 2014, 2015, 2016, 2017, 2018, 2019]

def mini(releve, date):
    temp_mini = releve[0]
    date_mini = date[0]
    for i in range(len(releve)):
        if releve[i] < temp_mini:
            temp_mini = releve[i]
            date_mini = date[i]
    return temp_mini, date_mini
```

" ) }}

## 0.16. Exercice 20.2 ☑

### Exercice 20.2

**Énoncé**

Un mot palindrome peut se lire de la même façon de gauche à droite ou de droite à gauche : *bob*, *radar*, et *non* sont des mots palindromes.

De même certains nombres sont eux aussi des palindromes : 33, 121, 345543.

L'objectif de cet exercice est d'obtenir un programme Python permettant de tester si un nombre est un nombre palindrome.

Pour remplir cette tâche, on vous demande de compléter le code des trois fonctions ci-dessous sachant que la fonction `est_nbre_palindrome` s'appuiera sur la fonction `est_palindrome` qui elle-même s'appuiera sur la fonction `inverse_chaine`.

La fonction `inverse_chaine` inverse l'ordre des caractères d'une chaîne de caractères `chaine` et renvoie la chaîne inversée.

La fonction `est_palindrome` teste si une chaine de caractères `chaine` est un palindrome. Elle renvoie `True` si c'est le cas et `False` sinon. Cette fonction s'appuie sur la fonction précédente.

La fonction `est_nbre_palindrome` teste si un nombre `nbre` est un palindrome. Elle renvoie `True` si c'est le cas et `False` sinon. Cette fonction s'appuie sur la fonction précédente.

Compléter le code des trois fonctions ci-dessous.

```
def inverse_chaine(chaine):
    result = ...
    for caractere in chaine:
        result = ...
    return result

def est_palindrome(chaine):
    inverse = inverse_chaine(chaine)
    return ...

def est_nbre_palindrome(nbre):
    chaine = ...
    return est_palindrome(chaine)
```

Exemples :

```
>>> inverse_chaine('bac')
'cab'
```

```
>>> est_palindrome('NSI')
False
>>> est_palindrome('ISN-NSI')
True
>>> est_nbre_palindrome(214312)
False
>>> est_nbre_palindrome(213312)
True
```

**Correction**

{{ correction(True, "

```
1  def inverse_chaine(chaine):
2      result = ''
3      for caractere in chaine:
4          result = caractere + result
5      return result
6
7  def est_palindrome(chaine):
8      inverse = inverse_chaine(chaine)
9      return chaine == inverse
10
11 def est_nbre_palindrome(nbre):
12     chaine = str(nbre)
13     return est_palindrome(chaine)
```

") }}

## 0.17. Exercice 29.1 ☑

### Exercice 29.1

#### Énoncé

Soit un nombre entier supérieur ou égal à 1 :

- s'il est pair, on le divise par 2 ;
- s'il est impair, on le multiplie par 3 et on ajoute 1.

Puis on recommence ces étapes avec le nombre entier obtenu, jusqu'à ce que l'on obtienne la valeur 1.

On définit ainsi la suite $(U_n)$ par :

- $U_0 = k$, où $k$ est un entier choisi initialement;
- $U_{n+1} = \dfrac{U_n}{2}$ si $U_n$ est pair;
- $U_{n+1} = 3 \times U_n + 1$ si $U_n$ est impair.

**On admet que, quel que soit l'entier `k` choisi au départ, la suite finit toujours sur la valeur 1.**

Écrire une fonction `calcul` prenant en paramètres un entier `n` strictement positif et qui renvoie la liste des valeurs de la suite, en partant de `n` et jusqu'à atteindre 1.

Exemple :

```
>>> calcul(7)
[7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

#### Correction

{{ correction(True, "

" ) }}

## 0.18. Exercice 29.2 ☑

### Exercice 29.2

**Énoncé**

On affecte à chaque lettre de l'alphabet un code selon le tableau ci-dessous :

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

Pour un mot donné, on détermine d'une part son *code alphabétique concaténé*, obtenu par la juxtaposition des codes de chacun de ses caractères, et d'autre part, *son code additionné*, qui est la somme des codes de chacun de ses caractères.

Par ailleurs, on dit que ce mot est « *parfait* » si le code additionné divise le code concaténé.

Exemples :

- Pour le mot `"PAUL"`, le code concaténé est la chaîne `'1612112'`, soit l'entier 1 612 112. Son code additionné est l'entier 50 car 16 + 1 + 21 + 12 = 50. 50 ne divise pas l'entier 1 612 112 ; par conséquent, le mot `"PAUL"` n'est pas parfait.
- Pour le mot `"ALAIN"`, le code concaténé est la chaîne `'1121914'`, soit l'entier 1 121 914. Le code additionné est l'entier 37 car 1 + 12 + 1 + 9 + 14 = 37. 37 divise l'entier 1 121 914 ; par conséquent, le mot `"ALAIN"` est parfait.

Compléter la fonction `est_parfait` ci-dessous qui prend comme argument une chaîne de caractères `mot` (en lettres majuscules) et qui renvoie le code alphabétique concaténé, le code additionné de `mot`, ainsi qu'un booléen qui indique si `mot` est parfait ou pas.

```
dico = {"A":1, "B":2, "C":3, "D":4, "E":5, "F":6, "G":7, \
"H":8, "I":9, "J":10, "K":11, "L":12, "M":13, \
"N":14, "O":15, "P":16, "Q":17, "R":18, "S":19, \
"T":20, "U":21,"V":22, "W":23, "X":24, "Y":25, "Z":26}

def est_parfait(mot) :
    #mot est une chaîne de caractères (en lettres majuscules)
    code_c = ""
    code_a = ???
    for c in mot :
        code_c = code_c + ???
        code_a = ???
    code_c = int(code_c)
```

```
    if ??? :
        mot_est_parfait = True
    else :
        mot_est_parfait = False
    return [code_a, code_c, mot_est_parfait]
```

Exemples :

```
>>> est_parfait("PAUL")
[50, 1612112, False]
>>> est_parfait("ALAIN")
[37, 1121914, True]
```

**Correction**

```
dico = {"A":1, "B":2, "C":3, "D":4, "E":5, "F":6, "G":7, \
"H":8, "I":9, "J":10, "K":11, "L":12, "M":13, \
"N":14, "O":15, "P":16, "Q":17, "R":18, "S":19, \
"T":20, "U":21,"V":22, "W":23, "X":24, "Y":25, "Z":26}

def est_parfait(mot) :
    #mot est une chaîne de caractères (en lettres majuscules)
    code_c = ""
    code_a = 0
    for c in mot :
        code_c = code_c + str(dico[c])
        code_a = code_a + dico[c]
    code_c = int(code_c)
    if code_c % code_a == 0:
        mot_est_parfait = True
    else :
        mot_est_parfait = False
    return [code_a, code_c, mot_est_parfait]
```