# Test de programmation

**Paul-Antoine BISGAMBIGLIA**
*Faculty of Science*
UMR-CNRS-6134
University of Corsica – 20250 Corte
*bisgambiglia@univ-corse.fr*

*Mots-clés : Algorithme, programmation*

## Exercice 1 :

Dans une chaîne, détecter la plus longue chaîne de caractères composés de caractères distincts.

Par exemple : “abcdemo” est la plus longue chaîne de caractères distincts de “abcdemoderneancien”

```
# Programme de test
def tester_plus_longue_sous_chaine_distincte():
    tests = [
        ("abcdemoderneancien", "abcdemo"),
        ("abcabcbb", "abc"),
        ("bbbbb", "b"),
        ("pwwkew", "wke"),
        ("", "")
    ]

    for chaine, resultat_attendu in tests:
        resultat_obtenu = plus_longue_sous_chaine_distincte(chaine)
        if resultat_obtenu == resultat_attendu:
            print(f'La plus longue sous-chaîne distincte de "{chaine}" est
: "{resultat_obtenu}"')
        else:
            print(f'Échec du test pour "{chaine}". Résultat attendu :
"{resultat_attendu}", Résultat obtenu : "{resultat_obtenu}"')

# Exécution du programme de test
tester_plus_longue_sous_chaine_distincte()
```

## Exercice 2 :

Concevoir un algorithme pour trouver tous les caractères communs à deux listes triées.

Par exemple, pour les listes [a, e, e, e] et [b, b, c, e, e, g], la sortie doit être de e, e.

## Exercice 3 :

Diviser un tableau de nombre en deux de manière à ce que la différence entre les deux tableaux soit le plus petit possible.

```
# Exemple d'utilisation :
liste = [3, 1, 4, 2, 2]
liste1, liste2 = diviser_liste_min_difference(liste)
print("Liste 1:", liste1)
print("Liste 2:", liste2)
```

## Exercice 4 :

Ecrire une fonction qui calcule la longueur moyenne des mots d'un texte.

```
sentence1 = "Même les phrases avec des caractères de la langue
française peuvent être utilisées."
sentence2 = "Le blog 'ledatascientist' est le blog français de
référence en Data Science."

print(average_words_length(sentence1)) [Output] => 5.38
print(average_words_length(sentence2)) [Output] => 5.17
```

## Exercice 5 :

Nous souhaitons inverser un entier (positif ou négatif), c'est-à-dire notre fonction prend en entrée un entier -6523 par exemple et retourne en sortie l'entier inversé -3256.

```
# Afficher les résultats de notre fonction pour 2020 et -9430
print(reverse_int(2020)) => 202
print(reverse_int(-9430)) => -349
```

## Exercice 6 :

Retrouver dans une liste d'entiers, tous les triplets pythagoriciens possibles qui y sont. Pour rappel, un triplet pythagoricien respecte le théorème suivant : $a^2 + b^2 = c^2$.
Prenons l'exemple suivant : nous avons la liste [0, 3, 6, 1, 2, 4, 5]. Notre fonction renvoie la liste des triplets possibles : [(3, 4, 5)] car 9 + 16 = 25.

## Exercice 7 :

Nous avons un mot et nous voulons savoir quel est le premier caractère unique de ce mot, c'est-à-dire la lettre qui ne se répète pas dans le mot et la première.

```
print(first_unique_character('coronavirus')) [Output] ('c', 0)
print(first_unique_character('Europe')) [Output]  ('u', 1)
```

## Exercice 8 :

Écrivez une fonction récursive somme_recursive qui prend en entrée une liste de nombres et retourne la somme de tous les éléments de la liste en utilisant la récursivité.

```
# Test de la fonction
liste = [1, 2, 3, 4, 5]
resultat = somme_recursive(liste)
print("La somme de la liste est :", resultat)
```

## Exercice 9 :

Écrivez une fonction qui teste si une chaine de caractères passée en paramètre est un palindrome : radar par exemple est un palindrome.

```
# Programme de test
def tester_est_palindrome():
    tests = [
        ("radar", True),
        ("Madam", True),
        ("A man a plan a canal Panama", True),
        ("hello", False),
        ("python", False),
    ]

    for chaine, resultat_attendu in tests:
        resultat_obtenu = est_palindrome(chaine)
        if resultat_obtenu == resultat_attendu:
            print(f'"{chaine}" est un palindrome : {resultat_obtenu}')
        else:
            print(f'"{chaine}" n\'est pas un palindrome. Résultat attendu :
{resultat_attendu}, Résultat obtenu : {resultat_obtenu}')

# Exécution du programme de test
tester_est_palindrome()
```

## Exercice 10 :

Écrivez une fonction qui cherche un élément dans une liste triée et qui se base sur la recherche dichotomique.

```
Algorithme RechercheDichotomique(liste, element):
    debut = 0
    fin = longueur(liste) - 1

    tant que début <= fin:
        milieu = (début + fin) div 2

        si liste[milieu] est égal à element:
            retourner milieu  # L'élément a été trouvé à l'index milieu.
        sinon si liste[milieu] < element:
            début = milieu + 1  # L'élément est dans la moitié droite de la
liste.
        sinon:
            fin = milieu - 1  # L'élément est dans la moitié gauche de la
liste.
```

```
    retourner -1  # L'élément n'est pas présent dans la liste.
```