Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2025-2026**

4. Modules externes - Casino (TP)

Exercice 4.6 *(Debugage d'une solution de l'exercice 3.15)*

L'équipe Python a écrit une solution pour l'exercice 3.15 sur le calcul du pdcg. Malheureusement le programme ne fonctionne pas ! Vous allez devoir le corriger et le faire fonctionner.

Pour récupérer le fichier, aller sur Moodle, dans la semaine 5, copier dans votre répertoire personnel, le fichier `Programme à corriger : Debug_pgcd.py`.

```
def pgcdParDiviseurs(a;b):
    '''calcul les diviseurs communs à a et b (avec a>b) et
    renvoie le plus grand diviseur commun'''
    i = 0
    while i < b:
        if a%i = 0 and b%i = 0 and i > pgcd:
            pgcd = i
        i = i+1
    print(pgcd)

def pgcdParDifferences(a;b):
    '''calcul le pgcd par l'algorithme des différences :
    si un nombre est un diviseur de 2 autres nombres a et b
    (avec a>b), alors il est aussi un diviseur
    de leur différence a - b'''
    diff = a - b
    while diff > 0:
        a = diff
        diff = a - b
    print(a)

def pgcdParEuclide(a;b)
    '''calcule le pgcd (avec a>b) par l'algorithme d'Euclide :
    renvoie le reste de la division euclidienne de a par b ;
    si le reste est égal à 0, alors le PGCD est égal à b,
    sinon a reçoit la valeur de b et b reçoit le reste
    et réitérer'''
    reste = a // b
    if reste != 0:
        a = b
        b = reste
        reste = a // b
        continue
    print(b)
```

```
print(pgcdParDiviseurs(36,15)
print(pgcdParDifferences(36,15)
print(pgcdParEuclide(36,15)
```

1. Exécuter le programme et corriger les erreurs de syntaxe annoncées par l'interpréteur.
2. Pourquoi le mot-clé `None` s'affiche-t-il ? Modifier le programme pour supprimer ces affichages inutiles et améliorer la lisibilité de l'affichage général.
3. Après avoir ré-exécuté le programme, que remarquez-vous ? Quel est le bon résultat ? Corriger les erreurs de conception algorithmique dans chacune des fonctions en ajoutant les affichages intermédiaires utiles.
4. Modifier le programme pour tester le cas $a = 40$, $b = 20$ et corriger si nécessaire.
5. Modifier le programme pour tester le cas $a = 4$, $b = 0$ et corriger si nécessaire.

Exercice 4.7

1. Écrire un programme Python qui permet de calculer et d'afficher le cosinus de $\pi/2$ (module `math`).
2. Écrire un programme Python qui permet d'afficher une série de 10 nombres réels aléatoires compris entre 10 et 50 (inclus) avec 1 seconde d'intervalle entre chaque écriture (modules `random` et `time`).

Exercice 4.8

1. Écrire une fonction Python qui aura deux arguments par mot-clé indiquant la borne inférieure et la borne supérieure d'un intervalle de tirage. Par défaut, l'intervalle de tirage sera $[0, 20]$. La fonction génèrera 10 valeurs entières aléatoires de l'intervalle et renverra la plus petite, la moyenne et la plus grande de ces valeurs.
2. Écrire le programme Python qui permet de tester votre fonction dans l'intervalle par défaut et d'afficher les 3 valeurs puis demande à l'utilisateur s'il veut changer les bornes par défaut et si oui quelles sont les bornes et les 3 valeurs associées.

Exercice 4.9

1. Écrire en Python une fonction `AleatoireAvecRepetitions(n)` qui prend en paramètre un entier strictement positif n et effectue le traitement suivant :
   — la fonction génère au hasard un nombre entier qui sera 0 ou 1. Elle répète ces tirages aléatoires jusqu'à avoir obtenu n fois 1 de façon consécutive ;
   — ensuite la fonction réalise un dernier tirage aléatoire entre 0 et 1 ;
   — la fonction doit alors renvoyer la valeur de ce dernier tirage aléatoire ainsi que le nombre total de tirages aléatoires qu'elle aura dû effectuer au préalable.
2. Écrire en Python le programme principal qui réalise les tâches suivantes :
   — le programme demande à l'utilisateur un nombre entier n strictement positif, correspondant au nombre de répétitions ;
   — le programme affiche ensuite le nombre de 1, renvoyés par 1000 appels à la fonction `AleatoireAvecRepetitions(n)`, et la valeur moyenne du nombre total de tirages.

3. Tester votre programme (après avoir testé votre fonction en demandant des affichages intermédiaires si nécessaires) pour des valeurs de n allant de 1 à 7. Que constatez-vous ? Comment l'expliquez-vous ?

Exercice 4.10 *(Évaluation de π par la méthode de Monte-Carlo)*

Soit un cercle, de centre $(0, 0)$ et de rayon $r = 1$, inscrit dans un carré de coté $l = 2$. L'aire du carré vaut 4 et l'aire du cercle vaut π. En choisissant N points aléatoires (à l'aide d'une distribution uniforme) à l'intérieur du carré, chaque point a une probabilité

$$p = \frac{\text{aire du cercle}}{\text{aire du carré}} = \frac{\pi}{4}$$

de se trouver également dans le cercle.

Soit n le nombre de points tirés aléatoirement se trouvant effectivement dans le cercle, on a :

$$p = \frac{n}{N} \approx \frac{\pi}{4}$$

d'où

$$\pi \approx 4 \times \frac{n}{N}$$

Déterminer une approximation de π par cette méthode. Pour cela, on procède en N itérations : à chaque itération, choisir aléatoirement les coordonnées d'un point entre -1 et 1 (fonction `uniform()` du module `random`), calculer la distance entre ce point et le centre du cercle, déterminer si cette distance est inférieure au rayon du cercle égal à 1, et si c'est le cas, incrémenter le compteur n de 1. Quelle est la qualité de l'approximation de π pour $N = 50$, $N = 500$, $N = 5\,000$ et $N = 50\,000$?

Exercice 4.11 *(À vous de jouer !)*

Écrire un programme de jeu de roulette (très simplifié) dans lequel le joueur peut miser une certaine somme sur un numéro et gagner ou perdre de l'argent si ce numéro est tiré aléatoirement. Le joueur peut enchainer plusieurs parties mais, il est contraint de s'arrêter s'il ne lui reste plus d'argent !

Voici la règle du jeu :

Le joueur mise une certaine somme sur un numéro compris entre 0 et 49 (50 numéros en tout) en déposant cette somme sur le numéro choisi sur la table de jeu : il donne donc sa mise au croupier. La roulette est constituée de 50 cases allant de 0 à 49. Les numéros pairs sont de couleur noire, les numéros impairs sont de couleur rouge. Le croupier lance la roulette, lâche la bille et quand la roulette s'arrête, relève le numéro de la case dans laquelle la bille s'est arrêtée. Le numéro sur lequel s'est arrêtée la bille est, naturellement, le numéro gagnant. Si le numéro gagnant est celui sur lequel le joueur a misé (probabilité de 1/50, plutôt faible), le croupier lui remet 3 fois la somme misée. Dans le cas contraire, si le numéro gagnant a la même couleur que celui sur lequel le joueur a misé, le croupier lui remet 1,5 fois la somme misée ; sinon, lorsque le joueur a misé sur le mauvais numéro de mauvaise couleur, le joueur perd définitivement sa mise.

Écrire dans un premier temps en pseudo-code l'algorithme conforme à la règle du jeu. Ensuite, traduire cet algorithme en Python. Dans votre programme, il faut lire différentes valeurs

(et vérifier leur cohérence vis-à-vis des valeurs attendues) : le budget initial du joueur, le numéro choisi par le joueur, le fait qu'il souhaite ou non continuer de jouer.

Remarque : avec cette règle du jeu, si le joueur mise sur la bonne couleur mais pas le bon numéro, son bénéfice est de 50% de la somme misée. Au bout de nombreuses parties, on risque d'arriver à des nombres flottants avec beaucoup de chiffres après la virgule. Alors autant arrondir au nombre supérieur. Ainsi, si le joueur mise 3 euros sur la bonne couleur mais pas le bon numéro, il gagne 2 euros et le croupier lui rend $3 + 2 = 5$ euros. Pour cela, on va utiliser une fonction du module `math` nommée `ceil()`.

Exercice 4.12

Lors d'une course hippique, il est possible de faire plusieurs types de pari dont :

- le pari simple qui consiste, soit, à parier sur le cheval "gagnant" (qui va arriver en première place), soit, à parier sur le cheval "placé" (qui va arriver dans les 3 premiers si au moins 8 chevaux sont inscrits à la course, ou dans les 2 premiers si entre 4 et 7 chevaux sont inscrits à la course),
- le "tiercé" qui consiste à parier sur les trois chevaux qui arriveront en tête de la course : le joueur gagne s'il a trouvé les trois chevaux gagnants dans l'ordre ou dans le désordre.

Il s'agit d'écrire un programme Python permettant de calculer les chances de gain d'un joueur et de simuler une course hippique.

1. Écrire les instructions Python permettant de lire le nombre `n` de chevaux partants, ce nombre étant saisi au clavier par l'utilisateur. Il faut obliger l'utilisateur à saisir une valeur valide : `n` doit être supérieur ou égal à 4 et inférieur ou égal à 20.
2. Continuer le programme de la question 1 en écrivant les instructions Python permettant de déterminer et d'afficher les chances de gain lors d'une course avec `n` chevaux au départ :
   - pour le pari simple : 1 chance sur `n` de trouver le cheval gagnant, 3 chances sur `n` de trouver le cheval placé s'il y a au moins 8 chevaux au départ de la course, ou 2 chances sur `n` de trouver le cheval placé s'il y a entre 4 et 7 chevaux inscrits à la course,
   - pour le tiercé, on a 1 chance sur A de le trouver dans l'ordre avec

     $$A = n \times (n-1) \times (n-2)$$

     et 1 chance sur $A/6$ de le trouver dans le désordre.

   Par exemple, supposons que 10 chevaux sont inscrits à la course, votre programme doit permettre d'afficher les informations suivantes :

   ```
   Vous avez 1 chance sur 10 de trouver le cheval gagnant
   Vous avez 3 chances sur 10 de trouver le cheval placé
   Vous avez 1 chance sur 720 de gagner le tiercé dans l'ordre
   Vous avez 1 chance sur 120 de gagner le tiercé dans le désordre
   ```

3. Continuer le programme de la question 2 en écrivant les instructions Python permettant de choisir aléatoirement l'ordre d'arrivée des trois chevaux gagnants dans une course hippique avec `n` chevaux au départ, numérotés de 1 à `n`. On rappelle que la fonction `randint(a,b)` du module `random` renvoie un nombre entier choisi aléatoirement dans l'intervalle [`a`, `b`] (avec `a` et `b` inclus). Attention à ce que les trois chevaux soient distincts !

4. Continuer le programme de la question 3 en écrivant les instructions Python permettant de simuler le pari d'un joueur. Pour cela, il faut lire le nombre `p` de chevaux sur lequel l'utilisateur souhaite parier. Il faut alors obliger l'utilisateur à saisir une valeur valide : 1 ou 3. Puis, les instructions doivent permettre de lire les numéros du ou des chevaux sur lesquels le joueur parie (on suppose alors que l'utilisateur ne fait pas d'erreur de saisie), et d'afficher le résultat du pari, à savoir l'une des 5 conclusions suivantes :

```
Bravo! Vous avez trouvé un cheval placé et il est gagnant
Bravo! Vous avez trouvé un cheval placé
Bravo! Vous avez trouvé le tiercé dans l'ordre
Bravo! Vous avez trouvé le tiercé dans le désordre
Perdu!
```

Exercice 4.13 *(Encore le calcul approché de $\pi = 3.1415926535897932...$)*

Considérer les suites ci-dessous :

— la suite de Leibniz : $u_n = 4 \times \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$

— la suite d'Euler : $v_n = \sqrt{6 \times \sum_{k=1}^{n} \frac{1}{k^2}}$

— la suite de Woon : $a_0 = 1$, $a_n = \sqrt{1 + \left(\sum_{k=0}^{n-1} a_k\right)^2}$ et $w_n = \frac{2^{n+1}}{a_n}$

— les Fractions Continues : $z_1 = 2 + \frac{2}{1+1}$, $z_2 = 2 + \frac{2}{1+\frac{1}{\frac{1}{2}+1}}$, $z_3 = 2 + \frac{2}{1+\frac{1}{\frac{1}{2}+\frac{1}{\frac{1}{3}+1}}}$,

$z_4 = 2 + \frac{2}{1+\frac{1}{\frac{1}{2}+\frac{1}{\frac{1}{3}+\frac{1}{\frac{1}{4}+1}}}}$ ...

1. Écrire, pour chaque suite, une fonction Python permettant de déterminer et de renvoyer le rang n à partir duquel le n-ième terme constitue une approximation de π avec une précision de 10^{-6}.
2. Reprendre votre programme de l'exercice 3.6. Transformer ce programme en fonction monteCarlo() qui prend un argument par mot-clé N et renvoie la valeur approximée de π par la méthode de Monte-Carlo. Pour une même valeur de N, appeler votre fonction 10 fois et calculer la valeur moyenne renvoyée. Donner les approximations obtenues de π pour $N = 100$, $N = 1000$ et $N = 10000$.
3. Écrire un programme principal qui appelle vos différentes fonctions et affiche les résultats selon le format ci-dessous :

```
Suite                 rang     approximation  temps de calcul
Suite de Leibniz      1000000  3.1415937      0.3276
Suite de Euler         954930  3.1415917      0.3083
Suite de Woon              11  3.1415923      0.0000
Fractions continues       627  3.1415917      0.0252
La valeur approximée de pi avec 100 points est 3.0920000
La valeur approximée de pi avec 1000 points est 3.1492000
La valeur approximée de pi avec 10000 points est 3.1470800
```

Pour calculer le temps de calcul il faut utiliser la fonction time du module time.