PSL **CPES 1re année**
Informatique **2023-2024**

TP4 : Exercices sur les fonctions

Debugger le programme de l'exercice 10 sur les instructions répétitives

Télécharger le fichier `Debug_instructionRepetitive.py` disponible sous moodle. Ce fichier contient un programme Python permettant de résoudre l'exercice 10 du TP sur les instructions répétitives. Malheureusement le programme ne fonctionne pas ! Vous allez devoir le corriger et le faire fonctionner. Vous commencerez par tester votre programme sur 3 valeurs : -1, 2, -4 ; puis, sur trois autres valeurs : 2, -1, 6 ; et enfin sur 5 valeurs, 1, 2, 3, 4, 5.

Utilisation de fonctions Python existantes

La liste des fonctions Python existantes (dont `print()` et `input()` par exemple) est disponible à l'adresse suivante :
`https://docs.Python.org/3/library/functions.html`

Dans l'interpréteur de commandes taper les instructions suivantes :

```
abs(-5)
help(abs)
max(5,12)
help(max)
min(5,12)
help(min)
```

Exercice 1

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 2. 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 3

Écrire :

1. une fonction `cube()` qui calcule le cube d'un entier n passé en argument ;
2. une fonction `volume()` qui calcule le volume d'une sphère de rayon r passé en argument en appelant la fonction `cube()` précédente. Pour rappel : le volume d'une sphère se calcule par la formule : $V = (4/3) \times \pi \times r^3$;
3. écrire un programme qui permet d'afficher le volume d'une sphère dont le rayon est saisi par l'utilisateur.

Exercice 4

Écrire en Python une fonction permettant d'afficher une table de multiplication avec 3 arguments : `base`, `debut` et `fin`, la fonction affichant le résultat de la multiplication de `base` par tous les entiers situés entre `debut` et `fin`. Par exemple l'appel de `tableMulti(8, 13, 17)` devra afficher :

```
Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
16 x 8 = 128
17 x 8 = 136
```

Exercice 5

Écrire 3 fonctions qui permettent de calculer le Plus Grand Commun Diviseur (PGCD) de deux nombres a et b (avec $a \geq b$) et l'algorithme permettant de les tester. [1]

1. La première fonction `pgcdParDiviseurs(a,b)`, avec $a \geq b$, calculera les diviseurs communs à chacun des deux nombres et renverra le plus grand diviseur commun (on pourra ne calculer que les diviseurs nécessaires).

   Par exemple, si on appelle `pgcdParDiviseurs(63,42)`, les diviseurs de `63` étant `1;3;7;9;21;63` et ceux de `42` étant `1;2;3;6;7;14;21;42`, on doit pouvoir afficher, à l'aide de la valeur renvoyée :

   ```
   Le plus grand diviseur commun de 63 et 42 est : 21
   ```

2. La deuxième fonction `pgcdParDifferences(a,b)`, avec $a \geq b$, utilisera l'algorithme des différences pour renvoyer le PGCD. Si un nombre est un diviseur de 2 autres nombres a et b, alors il est aussi un diviseur de leur différence $a - b$. Le PGCD de a et b est donc aussi celui de b et $a - b$.

   Par exemple, calculons le PGCD de `60` et `36`.

   `60 - 36 = 24`, donc le PGCD de `60` et `36` est un diviseur de `24`. On recommence en effectuant une nouvelle soustraction entre le résultat obtenu à la soustraction précédente (24) et le plus petit des deux termes de la soustraction précédente (36) : on obtient `36 - 24 = 12`. Par conséquent, le PGCD de `60` et `36` est un diviseur de `12` et on peut ré-appliquer la soustraction. On arrête l'algorithme dès que la soustraction donne un résultat nul. Le PGCD correspond au résultat obtenu à l'étape précédente. Voici l'illustration des étapes :

   ```
   60 - 36 = 24
   36 - 24 = 12
   24 - 12 = 12
   12 - 12 = 0
   ```

   Ainsi, l'appel de `pgcdParDifferences(60,36)` doit renvoyer 12.

3. La troisième fonction `pgcdParEuclide(a,b)`, avec $a \geq b$, utilisera l'algorithme d'Euclide pour renvoyer le PGCD. L'algorithme d'Euclide pour calculer le PGCD de deux entiers a et b (avec $a \geq b$) consiste à effectuer une suite de divisions euclidiennes :
   — étape 1 : effectuer la division euclidienne de a par b ; on obtient r le reste ; si r est égal à 0, alors fin et le PGCD est égal à b, sinon aller à l'étape 2 ;
   — étape 2 : a reçoit alors la valeur de b, b reçoit la valeur de r et aller à l'étape 1.

   Cette méthode en général est plus rapide.

   Voici l'illustration des étapes :

   ```
   561 divisé par 357 donne 1 en quotient et 204 en reste
   donc 561 = 357 x 1 + 204
   357 divisé par 204 donne 1 en quotient et 153 en reste
   donc 357 = 204x1 + 153
   204 divisé par 153 donne 1 en quotient et 51 en reste
   donc 204 = 153x1 + 51
   153 divisé par 51 donne 1 en quotient et 0 en reste
   ```

1. inspiré de http://www.mathematiquesfaciles.com/calculer-le-pgcd-d-un-nombre-avec-cours_2_75657.htm

```
donc 153 = 51x3+ 0
Le plus grand diviseur commun de 561 et 357 est : 51
```

Ainsi, l'appel de `pgcdParEuclide(561,357)` doit renvoyer 51.

Exercice 6

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 retourner 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 ;
   — le programme affiche alors 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 7 : 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. Ecrire, pour chaque suite, une fonction Python permettant de déterminer le rang n à partir duquel le n-ième terme constitue une approximation de π avec une précision de 10^{-6}. Vos fonctions doivent renvoyer deux valeurs : le rang n et l'approximation de π obtenue.
2. Reprendre votre programme de l'exercice 2 et transformer ce programme en une fonction appelée `monteCarlo()` qui prend un argument le nombre de points N et renvoie la valeur approximée de π par la méthode de Monte-Carlo.

3. Ecrire une fonction `simulation()` qui prend deux arguments : le nombre de points N et un argument par mot-clé `nb` de valeur par défaut 10. Cette fonction appelle votre fonction `monteCarlo()` 10 fois et renvoie la valeur moyenne renvoyée pour `N` points générés aléatoirement.
4. Ecrire 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.

Exercice 8 : Création de modules

Il est possible de créer des modules externes en regroupant certaines fonctions dans un fichier Python.

1. Écrire trois fonctions :
   — `fact(n)` qui renvoie la factorielle de l'entier passé en paramètre (supposé positif ou nul) ;
   — `binome(n, p)` qui renvoie le coefficient binomial :
   $$\binom{n}{p} = \frac{n!}{p!\,(n-p)!}$$
   — `triangle(n)` qui affiche sur une même ligne les coefficients binomiaux de la nieme ligne du triangle de Pascal.
2. Enregistrer ces trois fonctions dans un même fichier appelé `myMath.py`
3. Créer, dans le même répertoire, un autre fichier `calculs.py`. Dans ce fichier, écrire un programme python, qui va utiliser les fonctions définies dans `myMath.py`, pour afficher la factorielle des nombres de 0 à 6 ainsi que les sept premières lignes du triangle de Pascal. Pour cela, il faut importer ces fonctions à l'aide de la commande `import myMath`.
4. Il est intéressant de pouvoir tester les fonctions définies dans un module. Cependant, si on ajoute directement dans le code du module des appels aux fonctions, les lignes correspondantes seront exécutées lors de l'importation, ce qui n'est pas souhaitable.
   — Ajouter, directement dans `myMath.py`, l'affichage de $5!$, de $\binom{5}{3}$ et de la 3ieme ligne du triangle et exécuter à nouveau `calculs.py` en relançant l'interpréteur (pour forcer la relecture du module).
   — On voit que le résultat des trois tests s'affiche. Pour éviter cela, il faudrait que les tests ne soient exécutés que lorsque le module est exécuté et pas le fichier `calculs.py` (qui utilise ce module). Pour ce fait, il faut mettre les trois tests dans un bloc qui n'est exécuté que si le module est le fichier principal, condition qui peut être testée avec une ligne

```
if __name__ == "__main__":
```

— On peut maintenant vérifier que l'exécution directe du module provoque l'exécution des tests alors que son importation dans un autre fichier ne le fait pas.