Mathématiques et programmation (enseignement de spécialité) - 1

Objectifs :

1. Revoir le cours d'arithmétique du point de vue algorithmique (algorithme d'Euclide et algorithme d'Euclide étendu)
2. Construire correctement une boucle par l'intermédiaire de la notion d'invariant de boucle qui est une sorte de raisonnement par récurrence
3. Faire la transition entre l'arithmétique et le calcul matriciel avec l'algorithme d'Euclide étendu

Partie A : Fonctions La notion de fonction joue un rôle important en mathématiques et en algorithmique.

Regardons sur un cas particulier une fonction "simple" qui calcule le déterminant de deux vecteurs donnés du plan, dans un premier temps on définit la fonction puis on l'exécute sur un cas particulier.

Faire les premiers exemples directement dans l'interpréteur avec IDLE (ou tout autre environnement)

```
>>> def det(x1,y1,x2,y2):
        return x1*y2 - x2*y1

>>> det(1,2,3,4)
-2
```

Ex 1 :

1. Définir une fonction qui calcule le produit scalaire de deux vecteurs donnés du plan $\overrightarrow{u}(x1, y1)$ et $\overrightarrow{v}(x2, y2)$. (Donner un nom significatif à votre fonction)
2. Exécuter la fonction pour $\overrightarrow{u}(1, 2)$ et $\overrightarrow{v}(3, 4)$

Ex 2 :

1. Définir une fonction en Python qui retourne vrai lorsque deux vecteurs donnés du plan $\overrightarrow{u}(x1, y1)$ et $\overrightarrow{v}(x2, y2)$ sont **colinéaires**
2. Définir une fonction en Python qui retourne vrai lorsque deux vecteurs donnés du plan $\overrightarrow{u}(x1, y1)$ et $\overrightarrow{v}(x2, y2)$ sont **orthogonaux**
3. Tester les deux fonctions sur plusieurs cas particuliers

Ex 3 :

Critiquer le résultat suivant et corriger les deux fonctions de l'exercice 2

```
>>> sontColineaires(0.1,1,0.3,3)
False
>>> det(0.1,1,0.3,3)
5.551115123125783e-17
```

Ex 4 :

Définir une fonction en Python qui retourne le coefficient directeur (ou pente) de la droite du plan définie par les deux points $A(x1, y1)$ et $B(x2, y2)$. Prévoir un test et afficher un message comme "non défini" lorsque les deux points ont la même abscisse

Partie B : Récursivité Un résultat important vu en cours d'arithmétique concernant le pgcd de deux entiers naturels est :

> $\text{pgcd}(a,0) = a$
> $\text{pgcd}(a,b) = \text{pgcd}(q,r)$ lorsque $a = bq + r$ avec $0 \leqslant r < b$
> Le pgcd étant alors le dernier reste non nul de cette suite de divisions euclidiennes

Or avec les langages comme Python dans le corps d'une fonction on peut appeler la fonction elle-même, on appelle ceci la récursivité, ce qui nous permet de traduire directement la propriété mathématique (ci-dessus) dans une fonction récursive en Python.

Dans le langage Python a % b donne le reste de la division euclidienne de a par b

```
>>> def pgcd(a,b):
        if b == 0:
                return a
        else:
                return pgcd(b,a%b)


>>> pgcd(70,42)
14
>>> pgcd(42,70)
14
```

Ex 5 :

1. Définir la division euclidienne de deux entiers de manière récursive
2. On définit pour un entier naturel non nul $n!$ par $n! = n \times (n-1) \times (n-2) .... \times 2 \times 1$. Définir une fonction récursive qui permet de calculer $n!$
3. Soit la suite récurrente définie par $u_n = 2u_{n-1} + 1$ pour $n \geqslant 1$ et $u_0 = 2$. Définir une fonction récursive qui permet de calculer u_n en fonction de n

Ex 6 :

Visualiser la suite des appels d'une fonction récursive sur le site de pythontutor (http ://pythontutor.com) par exemple la fonction `pgcd(a,b)`

Ex 7 :

Pour bien comprendre le paragraphe suivant on rappelle qu'une variable est une étiquette associée à une référence en mémoire (adresse mémoire)

Entrer dans l'interpréteur la suite des instructions suivantes, la fonction `vars()` montre le dictionnaire (entre accolades) associant les variables en mémoire et les références

Pour une variable comme m de type entier l'association donne directement la **valeur** de la variable en omettant la référence, cependant pour la fonction `pgcd(a,b)` l'association donne la référence (at 0x.....) En informatique les nombres de la forme 0x sont en hexadécimal

```
>>> def pgcd(a,b):
        if b == 0:
                return a
        else:
                return pgcd(b,a % b)


>>> m = 70
>>> n = 42
>>> pgcd(m,n)
14
>>> vars()
{'__package__': None, '__doc__': None, '__builtins__': <module 'builtins' (built-in)>, 'm': 70, '__spec__': None,
ame__': '__main__', '__loader__': <class '_frozen_importlib.BuiltinImporter'>, 'pgcd': <function pgcd at 0x11207b5
>>> m = 72
>>> n = 40
>>> pgcd(m,n)
8
>>> vars()
{'__package__': None, '__doc__': None, '__builtins__': <module 'builtins' (built-in)>, 'm': 72, '__spec__': None,
ame__': '__main__', '__loader__': <class '_frozen_importlib.BuiltinImporter'>, 'pgcd': <function pgcd at 0x11207b5
>>>
```

Partie C : Preuves de programme Nous avons vu en cours d'arithmétique que l'algorithme d'Euclide du calcul du pgcd de deux entiers naturels a et b s'obtient par une suite de divisions euclidiennes basée sur les propriétés :

> $\text{pgcd}(a, 0) = a$
> $\text{pgcd}(a, b) = \text{pgcd}(q, r)$ lorsque $a = bq + r$ avec $0 \leqslant r < b$
> Le pgcd étant alors le dernier reste non nul de cette suite de divisions euclidiennes

Ce qui donne comme programme Python (**à partir de maintenant on va utiliser l'éditeur de texte plutôt que l'interpréteur car on veut réutiliser nos programmes**)

```
def pgcd(a,b):
    r1 = a
    r2 = b
    while r2!= 0:
        temp  = r1
        r1 = r2
        r2 = temp % r1
    return r1
#------------------------------
#   Programme principal
#------------------------------
a = int(input("Entrez un entier a "))
b = int(input("Entrez un entier b "))

print("le pgcd de a = ",a," et de b = ",b," est ",pgcd(a,b))
```

A l'exécution du programme on obtient :

```
>>>
Entrez un entier a 42
Entrez un entier b 70
le pgcd de a =  42  et de b =  70  est  14
```

Le programme a l'air d'être juste.

Est ce qu'un programme est juste parce qu'il vérifie quelques cas particuliers ? Est ce qu'une propriété mathématique est vraie parce qu'elle est vraie sur quelques cas particuliers ?

Comment être sur que dans cette fonction

1. la boucle while s'arrête à un certain moment (preuve d'arrêt) ?
2. ce qui est retourné est **toujours** le pgcd de a et b ?

Nous allons faire une **preuve du programme** (qui ressemble à la **démonstration par récurrence**) et qui, non seulement répond positivement aux deux questions posées ci-dessus mais va jouer aussi un rôle heuristique dans la définition correcte de fonction dans laquelle il y a une boucle

De manière **abstraite** toute boucle est de la forme :

```
Tant que E(X)
    X ← f(X)
```

Autrement dit **une boucle "ressemble" à une suite récurrente très générale** où

1. X désigne le **vecteur** formé par toutes les variables qui sont dans la boucle
   Dans notre exemple X = (temp, r1,r2)
2. $E(X)$ une expression **booléenne** (qui est vraie ou fausse) formée à partir de X
   Dans notre exemple $E(X)$ est $r2 \neq 0$
3. f une fonction générale s
   Dans notre exemple $f(X) = f($ temp, r1, r2) = (r1 % r2, r2, r1)
   pour remplacer

   ```
   temp ← r1
   r1 ← r2
   r2 ← temp%r1
   ```

Or dans le cours de mathématiques la technique de démonstration par récurrence nous a été utile pour démontrer des propriétés des suites récurrentes comme la monotonie

En informatique au lieu de propriété on parle plutôt d'assertion comme une relation entre les valeurs des variables d'un algorithme

On cherche donc une assertion $I(X)$ telle que :

1. I(X) est vraie avant la boucle (initialisation)
2. si (I(X) **et** E(X)) est vraie alors I(f(X)) est vraie aussi (hérédité)

On dit alors que I est un **invariant de boucle**

Montrons que $I(X)$ définie par r2 = temp % r1 est un invariant de boucle

1. **Initialisation** : On pourrait initialiser la variable `temp` avant la boucle mais ici on va considérer que l'initialisation est le premier tour de boucle et en effet après le premier tour de boucle les valeurs de X sont (a , b , a % b)
2. **Hérédité** : Supposons que $I(X)$ et $E(X)$ est vraie et montrons que $I(f(X)$ l'est aussi :
   on suppose donc que les valeurs de X sont (v1,v2,v3) telle que v3 = v1 % v2 et v3 $\neq 0$ et montrons que $I(f(X)$ est vraie
   Or puisque v3 $neq0$ on peut calculer $f(X)$ = (v2,v3,v2 % v3) et **la troisième composante du vecteur est bien le reste de la division euclidienne de la première composante par la deuxième**

Montrons maintenant que $E(X)$ finit par devenir faux :

En effet on a vu dans le cours de mathématiques qu'une **suite décroissante d'entiers naturels finit par s'annuler à partir d'un certain rang**

Conclusion

Lorsque $E(X)$ devient faux la boucle s'arrête et on ne calcule plus $f(X)$

Les valeurs de X sont en sortie de boucle (v1,v2,v3) avec v3 = v1 % v2 = 0

Or pgcd(a,b) = pgcd(v2,0) = v2 qui est la valeur contenue dans la variable r1 à la sortie de boucle voilà pourquoi on retourne r1

Nous avons donc prouvé que la boucle réalise effectivement ce qu'elle est censé faire pour a et b entiers naturels

Nous allons maintenant changer de façon de faire et partir de l'invariant de boucle vers la boucle c'est à dire se servir de la notion d'invariant de boucle pour construire la boucle

Ex 8

Vérifier que les deux termes consécutifs de la suite de Fibonnacci 11349903170 et 701408733 sont premiers entre eux

Partie D : Algorithme d'Euclide étendu Nous avons vu dans le cours d'arithmétique le théorème de Bezout qui stipule que le pgcd de a et b peut s'écrire comme une combinaison linéaire de a et b

L'algorithme d'Euclide retourne un vecteur composé du pgcd et des coefficients entiers de la combinaison linéaire

Pour trouver cet algorithme nous allons chercher d'abord l'invariant de boucle.

Que voulons nous ?

Nous souhaitons trouver une combinaison linéaire de a et b qui est égale au pgcd de a et b autrement dit $au + bv = d$ avec u et v des entiers relatifs

Or l'algorithme d'Euclide retourne $r1$ le pgcd de a et b , il suffit donc de maintenir constant r1 comme combinaison linéaire de a et b sous la forme

$r1 = au1 + bv1$

Mais on doit garder $r2$ et temp avec l'invariant r2 = temp % r1 pour obtenir le pgcd donc ...

Ex 9

1. $r1 = au1 + bv1$ est un morceau de l'invariant de boucle trouver l'autre morceau
2. Construire la boucle
3. Rédiger la preuve
4. Appliquer l'algorithme à 11349903170 et 701408733

Ex 10

Définir une fonction récursive pour l'algorithme d'Euclide étendu

Partie E : Algorithme d'Euclide étendu avec calcul matriciel La suite de Fibonnacci est définie par $F_n = F_{n-1} + F_{n-2}$ pour $n \geqslant 2$ avec $F_0 = 0$ et $F_1 = 1$

On remplace les deux combinaisons linéaires

$F_n = 1 \times F_n + 0 \times F_{n-1}$

$F_{n+1} = 1 \times F_n + 1 \times F_{n-1}$

par $X_n = AX_{n-1}$ où $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$

Par conséquent par récurrence on obtient $X_n = A^n X_0$ avec $X_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

Ex 11

Ouvrir Edupython pour pouvoir utiliser la bibliothèque numpy de Python

1. Justifier que le programme suivant (la boucle) calcule bien A^{10}
2. Ce programme fait afficher toutes les puissances intermédiaires qu'observez vous ?
3. Quelle instruction ajouter pour faire afficher F_{10} ?
4. Combien y -a-t-il eu de multiplications de matrices pour calculer A^{10} avec la boucle de ce programme ?

   Proposer une méthode pour calculer A^{10} en faisant moins de multiplications de matrices

```
import numpy as np

X0 = np.array([[0],[1]])
A = np.array([[0,1],[1,1]])
P = np.eye(2)
for i in range(10):
    P = P.dot(A)
    print(P)
```

On revient sur l'algorithme d'Euclide étendu

Nous avons vu que :

si les variables sont (r1,r2,u1,u2,v1,v2) alors après un tour de boucle on a (r2,r1 % r2,u2,u1-(r1 // r2)*u2,v1-(r1 // r2)*v2)

Matriciellement $\begin{pmatrix} u1 \\ u2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -r1//r2 \end{pmatrix} = \begin{pmatrix} u1 \\ u2 \end{pmatrix}$

Ex 12

Définir une fonction python qui retourne le vecteur (d,u,v) tel que $au + bv = d$ (Théorème de Bezout) en utilisant le calcul matriciel (voir ci-dessus)