Série 11 – algorithmes récurrents

Exercice n°1.

Ecrire un module qui calcule la somme suivante :

i. S1 = 1 + 2 + 3 + … + n

ii. S2= 1+1/2+1/3+……………….1/n

iii. S3=1+2 ! /3 + 3 !/5 + 4 !/7 + 5 !/9+ ……n !/2n-1

iv. S4 = 0 - 1 + 2 - 3 + 4 – 5 + 6 – 7 … …-+n

v. S5 $=1-\frac{1}{3^2}+\frac{1}{5^2}-\frac{1}{7^2}+\dots$

Exercice n°2.

Soient **x** et **a** deux réels donnés strictement positifs.
On se propose de calculer la somme **S** définie par la formule suivante :

$$\mathbf{S} = \mathbf{x} - \frac{\mathbf{x}^3}{\mathbf{a}^2} + \frac{\mathbf{x}^5}{\mathbf{a}^4} - \frac{\mathbf{x}^7}{\mathbf{a}^6} + \frac{\mathbf{x}^9}{\mathbf{a}^8} - \dots$$

Exercice n°3.

Soient n un entier naturel non nul. Ecrivez un programme permettant de calculer puis d'afficher la somme suivante :

$$\mathbf{Y} = \mathbf{C_n^0} - \mathbf{C_n^1} + \mathbf{C_n^2} - \mathbf{C_n^3} + \cdots + (-1)^n \mathbf{C_n^n}$$

Exercice n°4.

Soient a et b deux réels et n un entier naturel non nul. Ecrivez un programme permettant de vérifier la formule du binôme exprimée par l'égalité suivante :

$$(\mathbf{a}+\mathbf{b})^n = \mathbf{C_n^0 a^n} + \mathbf{C_n^1 a^{n-1} b^1} + \mathbf{C_n^2 a^{n-2} b^2} + \cdots + \mathbf{C_n^{n-1} a^1 b^{n-1}} + \mathbf{C_n^n b^n}.$$

Exercice n°5.

Nous proposons calculer une valeur approchée de π en utilisant la formule de Wallis :

$$\frac{\pi}{2} = \frac{2}{1} * \frac{2}{3} * \frac{4}{3} * \frac{4}{5} * \frac{6}{5} * \frac{6}{7} * \dots$$

Donner l'algorithme puis le script python de la fonction Calc_pi() permettant de calculer une valeur approchée de π en utilisant la formule de Wallis à 10^{-4} près

Exercice n°6.

Sachant que

$$\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \dots$$

Pour x très proche de zéro , écrire un algorithme qui permet de calculer cos(x) en utilisant la formule ci-dessus. Le calcul s'arrête quand la différence entre deux termes consécutifs devient <=10^{-4}. Le dernier terme calculé est une valeur approchée de cos(x)

Exercice n°7.

Faire le programme python permettant de calculer une valeur approchée de **e** à $\mathbf{10^{-4}}$ près, en utilisant la formule suivante :

$$e = 1 + \frac{1}{1} + \frac{1}{1\times 2} + \frac{1}{1\times 2\times 3} + \frac{1}{1\times 2\times 3\times 4} + \cdots = \sum_{n=0}^{+\infty} \frac{1}{n!}.$$

Exercice n°8.

Soit la formule suivante de **pi** :

$$\pi = 2*\sqrt{3}*\left(1-\frac{1}{3}*\frac{1}{3^1}+\frac{1}{5}*\frac{1}{3^2}-\frac{1}{7}*\frac{1}{3^3}+\dots\dots\dots\dots\dots\dots\right)$$

Etablir l'algorithme du module qui permet de calculer la valeur approchée de π à $\mathbf{10^{-4}}$ prés.
Le calcul s'arrête quand la différence entre deux termes consécutifs devient inférieure ou égale à 10^{-4}.

Exercice n°9.

Pour calculer une valeur approchée de la constante π on peut utiliser la formule **zêta de Riemann** suivante :

$$\frac{\Pi^2}{6}=\frac{2^2}{2^2-1}*\frac{3^2}{3^2-1}*\frac{5^2}{5^2-1}*\frac{7^2}{7^2-1}*\frac{11^2}{11^2-1}*\dots$$

Sachant que 2,3,5,7,11, … sont des nombres premiers

On demande de faire un module en python nommé **calcul_pi()** qui calcule et une valeur approchée de π à $\mathbf{10^{-6}}$ près en utilisant la formule de zêta de Reimann.

Exercice n°10.

1- Soit la formule suivante qui donne une valeur approchée de cos(x):

$$\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\dots=\sum_{k=0}^{\infty}\frac{(-1)^k x^{2k}}{(2k)!}$$

2- On demande de développer l'algorithme d'un module **calc_cos(x)** qui calcule une valeur approchée de cos(x) à 10^{-4} près.

Exercice n°11.

Ecrire l'algorithme d'un module permettant de calculer et d'afficher la valeur approchée de π à 10^{-4} près en utilisant la formule suivante :

$$\pi=\frac{2}{1!}+\frac{(1!)^2\,2^2}{3!}+\frac{(2!)^2\,2^3}{5!}+\frac{(3!)^2\,2^4}{7!}+\dots=\sum_{n=0}^{\infty}\frac{2^{n+1}(n!)^2}{(2n+1)!}$$

Avec **n!** = 1* 2 * 3 * 4* … * n

Le calcul s'arrête lorsque la différence entre deux termes consécutifs devient inférieure ou égale à 10^{-4}

Exercice n°12.

Ecrire un programme qui permet de calculer et afficher les N premiers termes d'une suite U définie par :

$$\begin{cases} \mathbf{U0 = 5} \\ \mathbf{Un = 2*Un\text{-}1 + 1.5} \end{cases}$$

Exercice n°13.

Soit la suite U définie par :

$$\begin{cases} U_0=1 \\ U_n=U_{n-1}/(1+U_{n-1}^2) \text{ pour tout } n\geq 1. \end{cases}$$

Ecrire un module itératif qui permet calculer et afficher les n premiers termes de la suite U.

Exercice n°14.

On se donne un entier naturel non nul **U0**, on se propose de construire la séquence d'entier (Un) définie par :
i>=0, Ui+1 est la somme des carres des chiffres de Ui

Exemple : si Ui= 423 alors Ui+1=$4^2+2^2+3^2$=29

Exercice n°15.

Ecrire une analyse, un algorithme qui permet de calculer un terme d'indice n de la suite ROBINSON définie par : Ui=a alors Ui+1= apparition de chaque chiffre dans apparait dans Ui

Exemple

Si U0=1 alors
U1 = 11 ''1 Se répète 1 fois dans U0''
U2=21 ''1 Se répète 2 fois dans U1''
U3=1211 '2 Se répète 1 fois et 1 se répète 1 fois dans U2''
U4=3112 etc --
U5=132112 --

Exercice n°16.

Ecrire un algorithme qui permet de calculer le nème terme de la suite définit par :

U_0 peut-être 0 ou 1

Pour définir les autres termes on remplace à chaque fois ''o'' par ''01'' et ''1'' par ''10''

Exemple

$U_0 = 0 \quad U_1 = 01 \quad U_2 = 0110 \quad U_3 = 01101001$ ....

Exercice n°17.

Soit la suite (U) définie par:

$$\begin{cases} \mathbf{U0= 2, U1= 3} \\ \mathbf{Un = Un\text{-}1+ 2*Un\text{-}2 \text{ pour tout n} >= 2} \end{cases}$$

En supposant que cette suite est croissante, écrire un programme permettant de lire un entier x (x > 2), de vérifier et d'afficher s'il est un terme de la suite U ou non. Dans l'affirmative afficher son rang.

Exercice n°18.

Soient a et b deux réels supérieurs ou égaux à 1. On considère la suite numérique définie par U_0= a ; U_1 = b, Et pour tout entier naturel n $U_{n+2} = \sqrt{U_n} + \sqrt{U_{n+1}}$

Ecrire le programme d'une procédure qui permet de calculer et d'afficher la valeur de U_n pour des valeurs de a et b supérieure ou égales à 1 et p

Exercice n°19.

Soit la suite **U** définie par :

$$\begin{cases} U_0 = 1 + \frac{1}{m} \text{ (avec } \mathbf{m}\text{, un entier strictement positif)} \\ U_n = 1 + \frac{1}{U_{n-1}} \text{ pour tout entier naturel } \mathbf{n \geq 1} \end{cases}$$

1) Ecrire **l'algorithme** d'une fonction intitulée **Calc_Suite** qui permet de calculer le **n**ième terme de cette suite pour un entier **m**. (**m** et **n** sont deux entiers saisis dans le programme appelant)

2) Quel est l'ordre de récurrence de cette fonction ? Justifier la réponse.

Exercice n°20.

Ecrire un programme qui permet de calculer puis d'afficher la racine carrée d'un réel positif x en utilisant la suite suivante:

$$\begin{cases} \mathbf{U0 = (1+x)/2} \\ \mathbf{Un+1 = (Un+ x/Un)/2} \end{cases}$$

Il s'agit de calculer les premiers termes de cette suite jusqu'a ce que la différence entre deux termes successifs devienne inferieur ou égale a 10^{-4}. Le dernier terme calcule est une valeur approchée de **√ x** à **10^{-4}** prés.

Exercice n°21.

En mathématiques, et plus particulièrement en combinatoire, les **nombres de Catalan** forment une suite d'entiers naturels utilisée dans divers problèmes de dénombrement. Cette suite est définie comme suit :

$$C \begin{cases} C_0 = 1 \\ C_{n+1} = \frac{(4*n+2)}{(n+2)} * C_n \qquad \forall \quad n+1>0 \end{cases}$$

Questions :

1- Ecrire un algorithme d'une fonction qui retourne le $n^{ème}$ terme de la suite.

Exercice n°22.

Soit **x** un réel positif et ***U*** une suite définie par :

$$\begin{cases} U_0 = \frac{1+x}{2} \\ U_n = \frac{1}{2}\left(U_{n-1} + \frac{x}{U_{n-1}}\right) \text{ pour tout } n > 0 \end{cases}$$

Le terme U_n est une valeur approchée de la racine carrée de **x** à **epsilon** près, si $\left|\frac{U_n - U_{n-1}}{U_{n-1}}\right| <$ **epsilon.**

Travail demandé :

1- Quel est l'ordre de récurrence de la suite ***U*** ? justifiez votre réponse.

2- Ecrire un algorithme d'une fonction **RacineU(x)** qui retourne une valeur approchée de la racine carrée d'un réel positif **x** à 10^{-4} près en utilisant la suite ***U*** définie précédemment.

N.B. *l'élève n'est pas appelé à saisir **x**.*

Exercice n°23.

Soient **k, n** deux entiers naturels et **U** une suite définie par : $\begin{cases} U_0 = 1 \\ U_{n+1} = k * U_n \end{cases}$

Questions :

1) Quel est l'ordre de récurrence de la suite **U** ? Justifiez votre réponse.

2) Calculer les termes U_1, U_2 et U_3 pour **k = 4**.

3) Parmi les trois propositions qu'elle qui correspond au rôle de la suite **U** :

- ✓ La suite **U** permet de calculer la factorielle de **k** (**k !**)
- ✓ La suite **U** permet de calculer k à la puissance **n** (**k^n**)
- ✓ La suite **U** permet de calculer le produit de **n** et **k** (**n*k**)

4) Ecrire un algorithme d'une fonction qui permet de calculer le terme U_n pour tout entier **n** supérieur ou égale à zéro (**n** est passé en paramètre).

Exercice n°24.

Soit la suite (Un) définie par :

$$\begin{cases} U_1 = Z1 \\ U_2 = Z2 \\ U_n = U_{n-2} + U_{n-1} \end{cases}$$

Telle que Z1 et Z2 sont deux nombres complexes.

Ecrire une analyse et en déduire l'algorithme d'un module qui permet de calculer et d'afficher les N premiers termes de cette suite en fonction de Z1, Z2 et N.

N.B:

- Un nombre complexe Z est composé d'une partie réelle (a) et une partie imaginaire (b) telles que : Z = a + i*b
- Soient Z1 = a1 + i*b1 et Z2 = a2 + i*b2
  ➔ La somme de Z1 et Z2 donne Z = (a1+a2) + i*(b1+b2).

Exercice n°25.

La suite de Frank est définie comme suit : $\begin{cases} U_1 = X \\ U_n = U_{n-1} + PGCD (N , U_{n-1}) \end{cases}$

Soit la suite $\mathbf{V_n = U_n - U_{n-1}}$ pour tout $\mathbf{n \geq 2}$. Les termes de la suite **V** sont soit égale à **1**, soit un nombre premier. Après le calcul d'un certain nombre de termes (Maximum 30 termes), la suite V est dite équilibrée si et seulement si le nombre des **1** est égal à celui des entiers premiers.

Exemples :

- ✓ Pour **X = 4**, le calcul des termes de la suite **V** donne :
  $\mathbf{V_2 = 2, V_3 = 3, V_4 = 1, V_5 = 5, V_6 = 3, V_7 = 1, V_8 = 1, V_9 = 1.}$
  ➔ Cette suite est équilibrée car le nombre des **1** est égal au nombre des entiers premiers.
- ✓ Pour **X = 7**, le calcul des **30 premiers termes** (de V_2 à V_{31}) de la suite **V** donne :
  1, 1, 1, 5, 3, 1, 1, 1, 1, 11, 3,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 23, 3, 1, 1, 1, 1, 1, 1, 1
  ➔ Cette suite n'est pas équilibrée car le nombre des **1** est différent du nombre des **entiers premiers** après le calcul de 30 termes de la suite **V**.

Travail à faire :

Ecrire un algorithme d'un module qui permet de déterminer si la suite **V** est équilibrée ou non.

N.B : X est donné en paramètre.

Exercice n°26.

Soient **A** et **B** deux entiers positifs et la fonction **F** définie comme suit :

$$\begin{cases} F(A,B) = 0 \; Si \; A = 0 \\ F(A,B) = F(A \; DIV \; 2, 2 * B) \; Si \; A \; est \; pair \\ F(A,B) = B + F(A \; DIV \; 2, 2 * B) \; Si \; A \; est \; impair \end{cases}$$

1- Calculer les valeurs de F (9, 3) et F (5, 5).

2- La fonction **F** permet de déterminer :

- ☐ le PPCM de deux entier A et B.
- ☐ le produit de deux entiers A et B.
- ☐ la puissance de deux entiers A et B.

3- Ecrire un algorithme récursif pour la fonction **F** (*A*, *B*).

Exercice n°27.

Pour évaluer a^n ($a^n = a * a * a ... * a$), avec a et n deux entiers naturels, on a besoin de **n−1** multiplications. En informatique, l'algorithme d'exponentiation rapide est un algorithme utilisé pour calculer rapidement des grandes puissances entières. Le principe de cet algorithme est basé sur le fait qu'on a :

$a^n = a^{n/2} * a^{n/2}$ lorsque n est pair et $a^n = a * a^{(n-1)/2} * a^{(n-1)/2}$ lorsque **n** est impair.

D'où :

$$a^n = \begin{cases} 1 & \text{si } n = 0 \\ a^{n/2} * a^{n/2} & \text{si n est pair} \\ a * a^{(n-1)/2} * a^{(n-1)/2} & \text{si n est impair} \end{cases}$$

Travail demandé :

Ecrire une fonction récursive puissance permettant de calculer a^n en utilisant le principe décrit précédemment.

Exercice n°28.

La suite de **Fibonacci** peut être définie comme suit :

$$\begin{cases} F_0 = 1 \\ F_1 = 1 \\ \text{Pour tout n pair, } F_n = (F_{p-1})^2 + (F_p)^2 & \text{avec } n = 2 * p \\ \text{Pour tout n impair, } F_n = (2*F_{p+1} - F_p)*F_p & \text{avec } n = 2 * p + 1 \end{cases}$$

a) Ecrire un algorithme d'une fonction récursive nommée **Fibo** qui permet de calculer le terme F_n de la suite de **Fibonacci**, en utilisant la suite **F** décrite précédemment.

b) La formule $S = F_{n+2} - 1$ permet de calculer la somme **S** des **n+1** premiers termes de la suite de **Fibonacci** (de F_0 à F_n).
En utilisant cette formule et la fonction **Fibo**, écrire un algorithme d'une fonction nommée **Fibo_Som** qui permet de calculer la somme S.

Exercice n°29.

Soit les suites U et V définies par :

$$\begin{cases} U_0 = 2\sqrt{3} \text{ et } V_0 = 3 \\ U_{n+1} = \frac{2U_nV_n}{U_n+V_n} \text{ pour tout } n \in N \\ V_{n+1} = \sqrt{U_{n+1} * V_n} \end{cases}$$

La suite V_n converge vers π.

Ecrire l'algorithme d'une fonction intitulée **Pi_archimed** qui permet de déterminer la valeur approchée de π à ***eps*** près. Il s'agit de calculer les premiers termes de cette suite jusqu'à ce que la différence entre deux termes successifs devienne inférieure ou égale à ***eps***. Le dernier terme calculé est une valeur approchée de π

Exercice n°30.

Pour calculer la racine carrée d'un réel positif **a**, on peut utiliser la suite U définit comme suit :

$$\begin{cases} U0 = \frac{(1+a)}{2} \\ Un = \frac{1}{2}(Un-1+\frac{a}{Un-1}) \end{cases}$$

Le terme Un est considéré la racine carrée de a lorsqu'il vérifie la condition suivante :

$\left| \frac{Un - Un-1}{Un-1} \right| <$ Epsilon

Ecrire l'algorithme d'un module qui permet de calculer la racine carrée d'un réel **a** comme décrit précédemment.

Exercice n°31.

Etant donnés **n** un entier strictement positif et $\mathbf{C_n^p}$ définie comme suit :

$$\mathbf{C_n^p} \begin{cases} \mathbf{C_n^0} = 1 \\ \mathbf{C_n^n} = 1 \\ \mathbf{C_n^p} = \mathbf{C_{n-1}^p} + \mathbf{C_{n-1}^{p-1}} \end{cases}$$

On se propose d'approcher la valeur de **S** définie par la formule suivante :

$$\mathbf{S = 1 + 2\,\frac{1}{3C_2^1} + 2^2\,\frac{1}{5C_4^2} + 2^3\,\frac{1}{7C_6^3} + \ldots}$$

Travail demandé :

1- En utilisant la définition donnée ci-dessus, écrire un algorithme d'une fonction nommée **Combinaison** permettant de calculer $\mathbf{C_n^p}$.

2- Utiliser la fonction **Combinaison** afin d'écrire un algorithme d'un module qui permet de déterminer une valeur approchée de **S** à **epsilon** près.