LES SUITES NUMÉRIQUES E02C

EXERCICE N°1 Relation de récurrence : premier contact (Le corrigé)

On donne la suite u définie pour par : $\begin{cases} u_0=7 \\ \forall n \in \mathbb{N} \, , \, u_{n+1} = 4u_n+7 \end{cases}$.

1) Identifier la fonction f du cours.

$f : x \mapsto 4x+7$

2) Déterminer, *si possible*, . u_1 , u_2 , u_8 et u_{1000}

▪ $u_1 = 4\times u_0+7 = 4\times 7+7$

$\boxed{u_1 = 35}$

▪ $u_2 = 4\times u_1+7 = 4\times 35+7$

$\boxed{u_2 = 147}$

▪ $u_8 = 4\times u_7+7 = \ldots$ heu ça va faire beaucoup de calculs !

On va utiliser la calculatrice.

▪ À l'aide la calculatrice

$\boxed{u_8 = 152915}$

▪ Il n'est pas (encore) possible (à ce stade du cours) de calculer u_{1000} dans un temps raisonnable.

C'était plus facile quand la suite était définie de manière explicite !

EXERCICE N°2 Suite et relation de récurrence : 2ème contact (Le corrigé)

On donne la suite v définie par : $\begin{cases} v_0 = 2 \\ \forall n \in \mathbb{N} \, , \, v_{n+1} = \dfrac{2v_n-2}{v_n-3} \end{cases}$.

(On admet que $\forall n \in \mathbb{N} \, , \, v_n \neq 3$ et donc que la suite est correctement définie)

1) Identifier la fonction f du cours.

$f : x \mapsto \dfrac{2x-2}{x-3}$

2) Déterminer v_1 , v_2 et v_{15} .

▪ $v_1 = \dfrac{2v_0-2}{v_0-3}$

$= \dfrac{2\times 2-2}{2-3}$

$\boxed{v_1 = -2}$

▪ $v_2 = \dfrac{2v_1-2}{v_1-3}$

$= \dfrac{2\times(-2)-2}{(-2)-3}$

$\boxed{v_2 = \dfrac{6}{5}}$

Pour v_{15} on utilise la calculatrice

▪ À l'aide la calculatrice

$\boxed{v_{15} \approx 0{,}44}$

EXERCICE N°3 Suite définie par un algorithme (Python) (Le corrigé)

On donne la suite $(w_n)_{n \in \mathbb{N}}$ définie par : $w_0 = 3$

Pour un terme w_n ,

w_{n+1} s'obtient de la façon suivante :
- ▫ Multiplier w_n par 2.
- ▫ Enlever 5 au résultat.

1) Écrire une fonction «premiers_termes_de_w» en Python qui prend comme argument un entier n et qui renvoie une liste contenant les valeurs des $n+1$ premiers termes de la suite.

```python
def premiers_termes_de_w(n):
    w = [3] #On déclare une liste avec un seul element w0
    for k in range(n):
        resultat = 2*w[k]  #1ere instruction de l'algorithme
        resultat = resultat - 5 #2eme instruction de l'algorithme
        w.append(resultat) # On "ajoute" le résultat à la liste w
    return w    #On renvoie la liste contenant les n+1 premiers termes

def premiers_termes_de_w_bis(n):
    w = [3] #On déclare une liste avec un seul element w0
    for k in range(n):
        w.append(2*w[k]-5) #on ajoute directement à la liste le résultat des deux instructions
    return w    #On renvoie la liste contenant les n+1 premiers termes

def premiers_termes_de_w_ter(n):
    w = [3] #On déclare une liste avec un seul element w0
    for _ in range(n):   #l'indice n'est pas utilisé donc on préfère "_" plutôt qu'une lettre
        w.append(2 * w[-1] - 5) # w[-1] est le dernier élélment (actuel) de la liste
    return w
```

2) Écrire une fonction « w » en Python qui prend comme argument un entier n et qui renvoie la valeur de w_{n+1} . (*On pourra utiliser la question 1*)

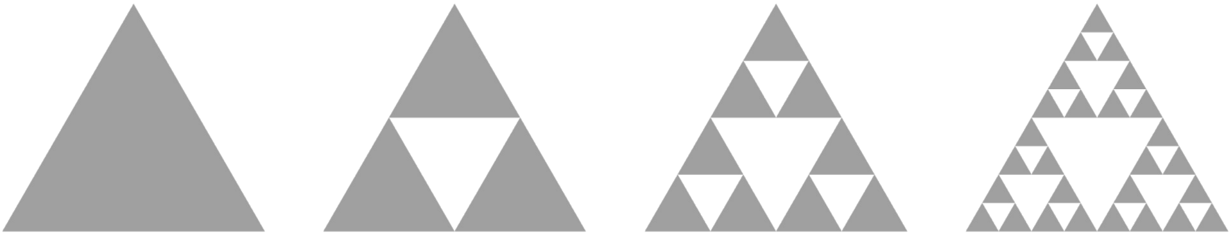
```python
def premiers_termes_de_w(n):
    w = [3] #On déclare une liste avec un seul element w0
    for k in range(n):
        resultat = 2*w[k]  #1ere instruction de l'algorithme
        resultat = resultat - 5 #2eme instruction de l'algorithme
        w.append(resultat) # On "ajoute" le résultat à la liste w
    return w[-1]    #On renvoie le dernier terme de la liste contenant les n+1 premiers termes

def premiers_termes_de_w_bis(n):
    w = [3] #On déclare une liste avec un seul element w0
    for k in range(n):
        w.append(2*w[k]-5) #on ajoute directement à la liste le résultat des deux instructions
    return w[-1]    #On renvoie le dernier terme de la liste contenant les n+1 premiers termes

def premiers_termes_de_w_ter(n):
    w = [3] #On déclare une liste avec un seul element w0
    for _ in range(n):   #l'indice n'est pas utilisé donc on préfère "_" plutôt qu'une lettre
        w.append(2 * w[-1] - 5) # w[-1] est le dernier élélment (actuel) de la liste
    return w[-1]
```

EXERCICE N°4 Triangle de Sierpinski (Le corrigé)

On considère un triangle équilatéral de côté 1 colorié en gris ($n=0$).
À chaque étape, on trace dans chaque triangle gris, un triangle blanc qui a pour sommets les milieux des côtés du triangle gris.



$n=0$ $n=1$ $n=2$ $n=3$

1) Il y a un triangle gris à l'étape 0, puis trois à l'étape 1…

1.a) Combien y-a-t-il de triangles gris, à l'étape 2 ?

Il y en a $\boxed{9}$.

1.b) Combien y-a-t-il de triangles gris, à l'étape 3 ?

Il y en a $\boxed{27}$.

1.c) Combien y-a-t-il de triangles gris, à l'étape 4 ?

On a remarqué que chaque triangles gris va en engendrer trois autres à l'étape suivante…

Il y en a $\boxed{81}$.

2) Pour tout $n \in \mathbb{N}$, on note u_n le nombre de triangles gris à l'étape n .

2.a) Exprimer u_{n+1} en fonction de u_n .

Pour tout $n \in \mathbb{N}$,

$\boxed{u_{n+1} = 3u_n}$

2.b) Exprimer u_n en fonction de n .

$u_0 = 1$

$u_1 = 3\times u_0$

$u_2 = 3\times u_1 = 3\times 3\times u_0$

$\vdots$

$u_n = 3\times u_{n-1} = \ldots = \underbrace{3\times 3\times \ldots\times 3}_{n\ fois}\times u_0 = 3^n\times u_0$ et comme $u_0 = 1$ …

Pour tout $n \in \mathbb{N}$,

$\boxed{u_n = 3^n}$

3) Déterminer le nombre de triangles gris à la 10e étape.

Il s'agit de calculer u_9 .

hé oui, on commence à $n=0$ ...

$u_9 = 3^9 = 19683$

Ainsi, $\boxed{\text{à la } 10^e \text{ étape, il y a } 19683 \text{ triangles gris}}$.

4) Déterminer u_{10} .

$u_{10} = 3^{10} = 59049$

$\boxed{u_{10} = 59049}$