Thème: Récursivité.

TI-83 Premium CE

Fibonacci - Le problème des chevaliers.

Introduction

On peut approcher la récursivité en prenant l'exemple d'une suite définie par récurrence :

Soit (u_n) la suite définie par $u_0 = 4$ et $\forall n \in \mathbb{N}\ u_{n+1} = 2u_n - 3$

Pour calculer u_n à l'aide d'une fonction Python on peut procéder ainsi :

ÉDITEUR : RECURSI
LIGNE DU SCRIPT 0001

```
def u(n):
  if n==0:
    return 4
  else:
    return 2*u(n-1)-3
```

PYTHON SHELL

```
>>>
>>> from RECURSI import *
>>> u(0)
4
>>> u(1)
5
>>> u(2)
7
>>> u(3)
11
```

On peut aussi utiliser un algorithme itératif :

ÉDITEUR : RECURSI
LIGNE DU SCRIPT 0017

```
def u2(n):
  u=4
  for i in range(n):
    u=2*u-3
  return u
```

PYTHON SHELL

```
>>>
>>> from RECURSI import *
>>> u2(0)
4
>>> u2(1)
5
>>> u2(2)
7
>>> u2(3)
11
```

Les fonctions définies de façon récursive ont un coup spatial qu'il ne faut pas ignorer.

Ainsi, sur la TI-83 Premium CE le nombre de récursions est limité à 25 :

PYTHON SHELL

```
>>> u(24)
16777219
>>> u(25)
33554435
>>> u(26)
  File "RECURSI.py", line 6, in
u
  File "RECURSI.py", line 6, in
u
  File "RECURSI.py", line 6, in
u
  File "RECURSI.py", line 6, in
u
RuntimeError: maximum recursion
depth exceeded
>>>
```

Fns... | a A # | Outils | éditer | Script

Pas de problème pour la définition avec itérations

PYTHON SHELL

```
>>> u2(700)
526013590154837350724098988288 01
286655503398028231738594982809 03
068732154297080822113666536277 58
845122698296885617821771301943 22
501838038631278147706518808499 55
223671128444598191663757884322 71
7271293251735781379
>>>
```

Fns... | a A # | Outils | éditer | Script

Thème: Récursivité.

TI-83 Premium CE

Fibonacci - Le problème des chevaliers.

python

Dans une script RECURSI

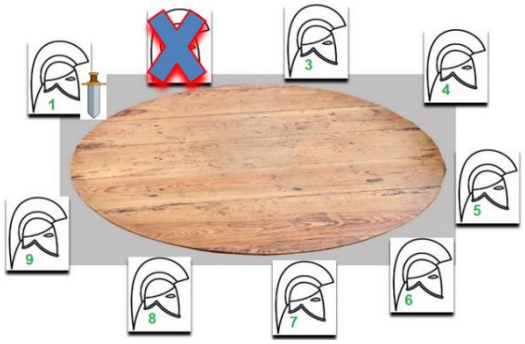
1°) Programmer le calcul de u_n par récursivité avec (u_n) la suite de Fibonacci.

On appellera `fibo` la fonction qui prend comme argument $n \in \mathbb{N}$ et qui renvoie u_n.

On rappelle que $u_0 = 0$ et $u_1 = 1$ et $\forall n \in \mathbb{N},\ n \geq 2 : u_n = u_{n-1} + u_{n-2}$

Application : Calculer u_{20}.

Dans un script KNIGHT



2°) N chevaliers sont disposés en cercle, portant chacun un numéro, 1,...,N. Le numéro 1 porte une épée avec laquelle il tue le chevalier à sa gauche (le numéro 2) et passe l'épée au numéro 3 qui tue le chevalier à sa gauche etc. L'épée tourne ainsi entre les chevaliers restants jusqu'à ce qu'il n'y ait plus qu'un seul chevalier debout.

Si on commence avec la liste suivante des chevaliers :

[1,2,3,4,5] (en rouge celui avec l'épée)

Les tours successifs seront :

[1,3,4,5]

[1,3,5]

[3,5]

[3]

La programmation de cet algorithme est plus simple si on place en premier le chevalier qui tient l'épée :

[1,2,3,4,5]

[3,4,5,1]

[5,1,3]

[3,5]

[3]

Soit `tue` la fonction qui prend en argument une liste de chevaliers et qui renvoie le chevalier survivant.

Compléter le script de la fonction `tue`.

Application : Quel est le chevalier vivant lorsqu'il y en a 5 au début ? Et s'il y en a 12 ?

```
ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0029

def tue(liste):
    if len(liste)==1:
        return liste[0]
    else:
        return tue(liste[2:]+liste[0])
```

Thème: Récursivité.

TI-83 Premium CE

Fibonacci - Le problème des chevaliers.

Fonction `fibo`

1°) Il faudra faire 3 cas : Les deux premiers étant utilisés pour le calcul de u_0 et de u_1.

L'autre cas fait un appel à `fibo(n-1)` et `fibo(n-2)`.

On trouve $u_{20} = 6765$.

ÉDITEUR : RECURSI
LIGNE DU SCRIPT 0023

```
def fibo(n):
  if n==0:
    return 0
  elif n==1:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
```

PYTHON SHELL

```
>>> fibo(20)
6765
```

Fonction `tue`

2°) En utilisant l'astuce de garder le chevalier à l'épée en tête de la liste, la récursivité est beaucoup plus facile.

Lorsqu'il reste un seul élément dans la liste des chevaliers, on le renvoie, c'est le vainqueur.

Sinon on reprend la liste de l'élément d'indice 2 jusqu'à la fin (car le chevalier d'indice 1 est mort). Et on recopie l'assassin, c'est-à-dire `liste[0]` à la fin de la liste.

ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0029

```
def tue(liste):
  if len(liste)==1:
    return liste[0]
  else:
    return tue(liste[2:]+[liste[
      0]])
```

PYTHON SHELL

```
>>> from KNIGHT import *
>>> tue([1,2,3,4,5])
3
>>> tue([1,2,3,4,5,6,7,8,9,10,11
,12])
9
```

On peut aussi programmer cette fonction de façon itérative, c'est un peu plus long, mais dans ce cas on peut prendre autant de chevaliers que l'on souhaite… (En Python le nombre de récursions est toujours limité).

ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0001

```
from math import *
def uncoup(liste,i):
  n=len(liste)
  if i==n-1:
    liste.pop(0)
    j=0
  else:
    liste.pop(i+1)
    j=(i+1)%(n-1)
  return liste,j
```

Fns... | a A # | Outils | Exéc | Script

ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0011

```
def chevalier(n):
  liste=list(range(1,n+1))
  i=0
  while len(liste)>1:
    tour=uncoup(liste,i)
    liste=tour[0]
    i=tour[1]
  return liste[0]

def tue(liste):
```

Fns... | a A # | Outils | Exéc | Script

On obtient les mêmes résultats que la fonction précédente.

PYTHON SHELL

```
>>> chevalier(5)
3
>>> chevalier(12)
9
```