

Structures répétitives [lp]

Exercices de cours

Karine Zampieri, Stéphane Rivière

Unisciel  algoprog  Version 17 mai 2018

Table des matières

1	Appréhender le cours	2
1.1	Table de multiplication / pgtmult	2
1.2	Entiers pseudo-aléatoires / pgalea	3
1.3	Lettre I / pglettreI	4
1.4	Triangle de chiffres / pgtriangle1	5
2	Appliquer le cours	7
2.1	Afficher les n premiers / pgaffich	7
2.2	Affichage systématique d'entiers / pgentiers	9
2.3	Multiples de 3 / pgmultiples3	11
2.4	Multiples de m / pgmultiples	13
2.5	Équerre de chiffres / pgequerre	14
2.6	Triangle de chiffres répétitifs / lptriangle3	16
3	Approfondir : Les suites	17
3.1	Le pas croissant / pgcroissant	17
3.2	La boîteuse / pgboiteuse	18
3.3	La suite de Fibonacci / pgfibonacci	19
3.4	La procession d'Echternach / pgechternach	20
3.5	La combinaison de deux suites / pgcmsuite	21
3.6	La capricieuse / pgcapricieuse	22

Python - Exercices de cours (Solution)



Mots-Clés Structures répétitives ■

Difficulté ●●○



Remarque

Les exercices **n'utilisent pas** le module @[Algorithmes paramétrés].

1 Appréhender le cours

1.1 Table de multiplication / pgtmult



Écrivez un script qui saisit un entier puis affiche sa table de multiplication. Exemple d'exécution :

```
n? 9
1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
...
10 * 9 = 90
```

Aide simple

Utilisez une boucle **Pour** de 1 à 10 dans laquelle affichez (où **j** est le compteur de boucle et **n** l'entier saisi) :

```
[j] * [n] = [j*n]
```



Testez.



Validez votre script avec la solution.

Solution Python

@[pgtmult.py]

```
def PGTmult():
    n = int(input("n? "))
    for j in range(1, 10+1):
        print(j, " * ", n, " = ", (j * n), sep="")
PGTmult()
```



Complétez votre script de sorte à faire la saisie de l'entier **jusqu'à** ce qu'il soit compris entre 1 et 9.

1.2 Entiers pseudo-aléatoires / pgalea



Définissez une constante entière `lim=10`.



Écrivez un script qui saisit un entier (dans `n`) puis affiche `n` entiers pseudo-aléatoires dans `[0..lim-1]`.

Outil Python

Le package `random` définit la fonction `randint(a,b)` qui renvoie un entier pseudo-aléatoire compris dans l'intervalle `[a..b]`.



Testez. Exemple d'exécution :

```
n? 8
0 7 3 1 2 3 5 9
```



De même, affichez `n` entiers pseudo-aléatoires dans `[-lim+1..lim-1]`.

Aide simple

Faites la différence de deux entiers pseudo-aléatoires dans `[0..lim-1]`.



Testez. Exemple d'exécution :

```
n? 10
8 4 6 1 0 7 4 6 8 9
-6 -9 1 0 -1 -6 2 5 -1 -2
```



Validez votre script avec la solution.

Solution Python

@[pgalea.py]

```
import random as rnd
lim = 10

def PGalea():
    n = int(input("n? "))
    for j in range(1, n+1):
        print(rnd.randint(0,lim-1), " ", sep="", end="")
    print()

    for j in range(1, n+1):
        print((rnd.randint(0,lim-1) - rnd.randint(0,lim-1)), " ", sep="", end="")
    print()

PGalea()
```

1.3 Lettre I / pglettreI



Écrivez un script qui saisit un entier (dans `n`) puis affiche la lettre I majuscule de hauteur `n` et largeur 3.

Résultat d'exécution :

```
n? 5
*****
***
***
***
*****
```



Testez.



Validez votre script avec la solution.

Solution Python

@[pglettreI.py]

```
def PGLettreI():
    n = int(input("n? "))
    print("*****")
    for j in range(n - 2, 1-1, -1):
        print(" ***")
    print("*****")

PGLettreI()
```

1.4 Triangle de chiffres / pgtriangle1



Écrivez un script qui saisit le nombre de lignes dans un entier `n`, puis affiche le triangle de **chiffres** de hauteur `n`. Affichez l'invite :

Nombre de lignes?

Outil

Un chiffre est un entier dans $[0..9]$, c.-à-d. que dans le cas où `n` est supérieur à 10, la séquence affichée est 1234567890123.... Utilisez le modulo pour cela.



Testez. Exemple d'exécution :

Nombre de lignes? 5

```
1
12
123
1234
12345
```



Validez votre script avec la solution.

Solution Python

@[pgtriangle1.py]

```
def PGTriangle1():
    n = int(input("Nombre de lignes? "))
    for j in range(1, n+1):
        for k in range(1, j+1):
            print(k % 10, end=" ")
        print()
PGTriangle1()
```



Modifiez votre script de sorte qu'il affiche le triangle gauche de **chiffres décroissants** de hauteur `n`.

Aide simple

La boucle sur les lignes est croissante et celle sur les colonnes est décroissante.



Testez.



Validez votre script avec la solution.

Solution Python @[pgtriangle2.py]

```
def PGTriangle2():
    n = int(input("Nombre de lignes? "))
    for j in range(1, n+1):
        for k in range(j, 1-1, -1):
            print(k % 10, end=" ")
        print()
PGTriangle2()
```

2 Appliquer le cours



Conseil

Vous devriez à présent être en mesure de résoudre les exercices qui suivent.
Courage!
Et n'en passez aucun !

2.1 Afficher les n premiers / `pgaffich`



Objectif

Cet exercice affiche les n premiers d'une séquence d'entiers.
Dans chacune des questions, outre la saisie, utilisez une boucle **Pour** : en effet :

- Soit le nombre de tours est connu.
- Soit les bornes et le pas de la boucle sont connus.



Écrivez un script qui saisit un entier dans n .
Affichez l'invite :

`n?`



Affichez les n premiers entiers strictement positifs :

`1 2 3...`

Rappel : Utilisez une boucle **Pour**.



Affichez les n premiers entiers strictement positifs en ordre **décroissant** :

`...3 2 1`



Affichez la séquence alternée jusqu'à n :

`1 -1 2 -2...`

Aide simple

Dans la boucle, il faut afficher le compteur et son négatif.



Affichez les entiers de $-n$ à n :

`...-2 -1 0 1 2...`



Affichez les n premiers naturels **impairs** :

`1 3 5 7...`



Affichez les naturels **impairs** qui sont **inférieurs ou égaux** à **n**.



Affichez les multiples de 5 compris entre 1 et **n**.



Affichez les multiples de **n** compris entre 1 et 100.



Validez votre script avec la solution.

Solution Python

@[pgaffich.py]

```
def PGAffich():
    n = int(input("n? "))

    for j in range(1, n+1):
        print(j, " ", sep="", end="")
    print()

    for j in range(n, 1-1, -1):
        print(j, " ", sep="", end="")
    print()

    for j in range(1, n+1):
        print(j, " ", (-j), " ", sep="", end="")
    print()

    for j in range(-n, n+1):
        print(j, " ", sep="", end="")
    print()

    for j in range(1, n+1):
        print((2 * j - 1), " ", sep="", end="")
    print()

    for j in range(1, n+1, 2):
        print(j, " ", sep="", end="")
    print()

    for j in range(5, n+1, 5):
        print(j, " ", sep="", end="")
    print()

    for j in range(n, 100+1, n):
        print(j, " ", sep="", end="")
    print()

PGAffich()
```


2.2 Affichage systématique d'entiers / pgentiers



Écrivez un script qui saisit la valeur de départ dans `vdep` (entier) et celle de fin dans `vfin` (entier) d'un intervalle. Affichez l'invite :

Valeur de départ et de fin?



Quelle est la boucle (`Itérer`, `Pour`, `TantQue`, `Répéter`) la plus adaptée pour afficher les entiers de l'intervalle `[vdep..vfin]` ?

Solution simple

La boucle `Pour` : les bornes et le pas de la boucle sont connus.



Affichez les entiers de cet intervalle `[vdep..vfin]` en ordre croissant.



Testez. Exemple d'exécution :

Valeur de départ et de fin? -3 5
-3 -2 -1 0 1 2 3 4 5



Validez votre script avec la solution.

Solution Python

@[pgentiers1.py]

```
def PGEntiers1():
    print("Valeur de depart et de fin? ", sep="", end="")
    vdep = int(input())
    vfin = int(input())
    for j in range(vdep, vfin+1):
        print(j, " ", sep="", end="")
    print()

PGEntiers1()
```



Que se passe-t-il si la valeur de départ `vdep` est supérieure à la valeur de fin `vfin` ? Vérifiez-le (si besoin).

Solution simple

Rien n'est affiché : on n'entre pas dans la boucle.



Modifiez votre script de sorte qu'il affiche systématiquement les entiers d'un intervalle donné. Exemple d'exécution :

Valeur de départ et de fin? 5 -2
5 4 3 2 1 0 -1 -2

Aide simple

Écrivez une boucle croissante ou décroissante selon que borne de départ `vdep` est inférieure ou supérieure à `vfin`.



Testez.



Validez votre script avec la solution.

Solution Python

@[pgentiers2.py]

```
def PGEntiers2():
    print("Valeur de depart et de fin? ", sep="", end="")
    vdep = int(input())
    vfin = int(input())
    if vdep <= vfin:
        for j in range(vdep, vfin+1):
            print(j, " ", sep="", end="")
    else:
        for j in range(vdep, vfin-1, -1):
            print(j, " ", sep="", end="")
    print()
PGEntiers2()
```

2.3 Multiples de 3 / pgmultiples3



Écrivez un script qui saisit une série d'entiers (supposés positifs) dans **nombre** jusqu'à ce que l'utilisateur tape la valeur 0.



Complétez la boucle afin d'afficher les nombres multiples de 3 au fur et à mesure.



Complétez votre code de sorte à compter le nombre de ces multiples dans un entier **n** (par exemple).

Solution simple

Il faut déclarer la variable, l'initialiser à zéro et l'actualiser dans le corps de la boucle si le nombre saisi est un multiple.



Affichez (où **[x]** désigne le contenu de **x**) :

Nombre de multiples : **[n]**



Testez. Exemple d'exécution :

Vos entiers. Tapez 0 pour finir

```
6
==> 6
5
4
9
==> 9
8
0
```

Nombre de multiples = 2



Validez votre script avec la solution.

Solution Python

@[pgmultiples3.py]

```
def PGMultiples3():
    n = 0
    print("Vos entiers. Tapez 0 pour finir")
    nombre = int(input("? "))
    while nombre != 0:
        if nombre % 3 == 0:
            print("==> ", nombre, sep="")
            n += 1
        nombre = int(input("? "))
    print("Nombre de multiples = ", n, sep="")
```

PGMultiples3()

2.4 Multiples de m / pgmultiples



Soient m et n deux entiers strictement positifs. Écrivez un script qui affiche les valeurs multiples de m inférieures à n en utilisant une structure itérative **Pour** avec un pas différent de 1.

Solution simple

Les multiples de m sont : 0, m , $2 \times m$, $3 \times m$ et ainsi de suite. Ainsi en partant de 0 (le premier multiple de m), le multiple suivant s'obtient à partir du multiple précédent en ajoutant m .

Nous ne disposons pas d'informations sur le nombre de multiples à afficher mais nous avons la valeur maximale, n , à ne pas dépasser. De plus, l'expression définissant le pas est la valeur de m .



Testez.



Validez votre script avec la solution.

Solution Python

@[pgmultiples.py]

```
def PGMultiples():
    n = int(input("Limite superieure? "))
    m = int(input("Valeur de reference"))
    print("Les multiples de ", m, " inferieurs a ", n, " sont:", sep="")
    for j in range(0, n+1, m):
        print(j, " ", sep="", end="")
    print()
PGMultiples()
```

2.5 Équerre de chiffres / pgequerre



Écrivez un script qui saisit le nombre de lignes dans un entier **n**, puis affiche l'équerre de **chiffres** de hauteur **n** (entier). (Un chiffre est un entier dans $[0..9]$, c.-à-d. que dans le cas où **n** est supérieur à 10, la séquence affichée est 1234567890123....) Exemple d'exécution :

```
Nombre de lignes? 5
12345
1234
123
12
1
```

Solution simple

On utilise une boucle imbriquée **Pour**. Sur la ligne 1, il y a **n** chiffres, sur la ligne 2, il y en a **n-1**... Par conséquent, la boucle sur les lignes est décroissante.



Testez.



Validez votre script avec la solution.

Solution Python

@[pgequerre1.py]

```
def PGEquerre1():
    n = int(input("Nombre de lignes? "))
    for j in range(n, 1-1, -1):
        for k in range(1, j+1):
            print((k % 10), end="")
        print()
PGEquerre1()
```



Modifiez votre script de sorte qu'il affiche l'équerre gauche de **chiffres décroissant** de hauteur **n**.

Solution simple

Sur la ligne 1, il y a **n** chiffres, sur la ligne 2, il y en a **n-1**... Par conséquent, la boucle sur les lignes est décroissante. La boucle sur les colonnes est également décroissante.



Testez.



Validez votre script avec la solution.

Solution Python

@[pgequerre2.py]

```
def PGEquerre2():  
    n = int(input("Nombre de lignes? "))  
    for j in range(n, 1-1, -1):  
        for k in range(j, 1-1, -1):  
            print(k % 10, end=" ")  
        print()  
  
PGEquerre2()
```

2.6 Triangle de chiffres répétitifs / lptriangle3



Écrivez un script qui saisit le nombre de lignes dans un entier `n` puis affiche le triangle de **chiffres répétitifs** de hauteur `n`. (Un chiffre est un entier dans $[0..9]$, c.-à-d. que dans le cas où `n` est supérieur à 10, la séquence affichée sera le **même dernier chiffre** de `j` (numéro de ligne). Exemple d'exécution.

```
Nombre de lignes? 5
1
22
333
4444
55555
```



Testez.



Validez votre script avec la solution.

Solution Python

@[pgtriangle3.py]

```
def PGTriangle3():
    n = int(input("Nombre de lignes? "))
    for j in range(1, n+1):
        for k in range(1, j+1):
            print(j % 10, end="")
        print()

PGTriangle3()
```



Que faut-il faire dans la boucle précédente pour afficher le triangle de chiffres suivant ?

```
Nombre de lignes? 5
5
44
333
2222
11111
```



Testez.



Quel est le résultat obtenu si dans votre code on fait une boucle `Pour` externe décroissante ?



Testez.

3 Approfondir : Les suites

3.1 Le pas croissant / pgcroissant



Objectif

Le pas croissant est la suite :

1, 2, 4, 7, 11, 16, ...

Comme on peut le constater, à chaque étape on ajoute un peu plus à l'entier précédent :

$$1 \xrightarrow{+1} 2 \xrightarrow{+2} 4 \xrightarrow{+3} 7 \xrightarrow{+4} 11 \xrightarrow{+5} 16 \dots$$



Écrivez un script qui saisit un entier dans `n` puis affiche les `n` premiers termes de la suite.



Validez votre script avec la solution.

Solution Python

@[pgcroissant.py]

```
def PGCroissant():
    n = int(input("n? "))
    nombre = 1
    pasRestants = n
    direction = 1
    for j in range(1, n+1):
        print(nombre, " ", sep="", end="")
        nombre += direction
        pasRestants -= 1
        direction += 1
    print()
PGCroissant()
```

3.2 La boiteuse / pgboiteuse



Objectif

La boiteuse est la suite :

1, 2, 4, 5, 7, 8, 10, 11, ...



Écrivez un script qui saisit un entier dans `n` puis affiche les `n` **premiers termes** de la suite.



Validez votre script avec la solution.

Solution Python

@[pgboiteuse.py]

```
def PGBoiteuse():
    n = int(input("n? "))
    nombre = 1
    pasRestants = 1
    direction = 1
    for j in range(1, n+1):
        print(nombre, " ", sep="", end="")
        nombre += direction
        pasRestants -= 1
        direction += 1
        if pasRestants == 0:
            direction = 2
            pasRestants = 2
    print()
PGBoiteuse()
```

3.3 La suite de Fibonacci / pgfibonacci



Objectif

La suite de Fibonacci est :

0, 1, 1, 2, 3, 5, 8, 13, 21, ...



Écrivez un script qui saisit un entier dans `n` puis affiche les `n` **premiers termes** de la suite.



Validez votre script avec la solution.

Solution Python

@[pgfibonacci.py]

```
def PGFibonacci():
    n = int(input("n? "))
    print(0, " ", sep="", end="")
    nombre = 1
    pasRestants = 1
    direction = pasRestants
    for j in range(1, n+1):
        print(nombre, " ", sep="", end="")
        pasRestants = nombre
        nombre += direction
        direction = pasRestants
    print()
PGFibonacci()
```

3.4 La procession d'Echternach / pgechternach



Objectif

La procession d'Echternach est la suite :

1, 2, 3, 4, 3, 2, 3, 4, 5, 4, 3, 4, 5, 6, 5, 4, 5, 6, ...



Écrivez un script qui saisit un entier dans `n` puis affiche les `n` **premiers termes** de la suite.



Validez votre script avec la solution.

Solution Python

@[pgechternach.py]

```
def PGEchternach():
    n = int(input("n? "))
    nombre = 1
    pasRestants = 3
    direction = 1
    for j in range(1, n+1):
        print(nombre, " ", sep="", end="")
        nombre += direction
        pasRestants -= 1
        if pasRestants == 0:
            direction = -direction
            pasRestants = (3 if direction == 1 else 2)
    print()
PGEchternach()
```

3.5 La combinaison de deux suites / pgcmsuite



Objectif

La combinaison de deux suites est :

1, 2, 3, 3, 5, 4, 7, 5, 9, 6, 11, 7, 13, 8, ...



Écrivez un script qui saisit un entier dans `n` puis affiche les `n` premiers termes de la suite.



Validez votre script avec la solution.

Solution Python

@[pgcmsuite.py]

```
def PGCmsuite():
    n = int(input("n? "))
    nombre = 1
    n1 = 1
    n2 = 1
    direction = 1
    for j in range(1, n+1):
        print(nombre, " ", sep="", end="")
        if direction == 1:
            n1 += 1
            direction = -1
            nombre = n1
        else:
            n2 += 2
            direction = 1
            nombre = n2
    print()
PGCmsuite()
```

3.6 La capricieuse / pgcapricieuse



Objectif

La **capricieuse** est la suite :

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,  
20, 19, 18, 17, 16, 15, 14, 13, 12, 11,  
21, 22, 23, 24, 25, 26, 27, 28, 29, 30,  
40, 39, 38, ..., 31,  
41, 42, ...
```



Écrivez un script qui saisit un entier dans **n** puis affiche les **n premiers termes** de la suite.



Validez votre script avec la solution.

Solution Python

@[pgcapricieuse.py]

```
def PGCapricieuse():  
    n = int(input("n? "))  
    nombre = 1  
    pasRestants = 9  
    direction = 1  
    for j in range(1, n+1):  
        print(nombre, " ", sep="", end="")  
        nombre += direction  
        pasRestants -= 1  
        if pasRestants == 0:  
            print(nombre, " ", sep="", end="")  
            direction = -direction  
            pasRestants = 9  
            nombre += 10  
    print()  
PGCapricieuse()
```