Chapitre 3

Algorithmes classiques

Nous allons présenter ici un certain nombre d'algorithmes classiques (nombres premiers, PGCD et PPCM, racines entières d'un polynôme, etc.), ce qui nous donnera l'occasion d'appliquer les bases de Python vues au chapitre précédent à l'algorithmique.

Arithmétique

AUTOUR DES NOMBRES PREMIERS

Théorème : Soit *d* et *n* deux entiers non nuls. *d* est un diviseur de *n* si et seulement si le reste de la division euclidienne de *n* par *d* est nul.

Objectif 1 Nous allons construire une fonction `diviseur` avec deux paramètres, *d*, un entier naturel non nul, et *n*, un entier naturel supérieur strict à 1, et qui renvoie `True` si *d* divise *n*, *False* sinon.

Algorithme

```
fonction diviseur(d,n):
   Si d divise n alors
      Renvoyer True
   Sinon
      Renvoyer False
   FinSi
Fin
```

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0013

```
def diviseur(d,n):
    if n%d==0:
        return True
    else:
        return False
```

Exécutons maintenant cette fonction en console.

2 est bien un diviseur de 10, mais pas 3.

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> diviseur(2,10)
True
>>> diviseur(3,10)
False
```

go.eyrolles.com/ti-python6

Objectif 2 Nous allons construire une fonction `nbdiviseur` avec un paramètre *n* entier naturel supérieur strict à 1 et qui renvoie le nombre de diviseurs positifs de n.

c va représenter notre compteur, c'est-à-dire le nombre de diviseur de *n*. 1 étant un diviseur de *n*, on va initialiser *c* à 1 et tester si *d* est un diviseur de *n* pour *d* allant de 2 à *n*.

Algorithme

```
fonction nbdiviseur(n):
   c←1
      Pour d allant de 2 à n
         Si diviseur(d,n)=True alors
            c←c+1
         FinSi
      FinPour
   Renvoyer c
Fin
```

Déterminons le nombre de diviseurs de 10, puis de 30 et de 31.

10 admet 4 diviseurs, 30 en admet 8 et 31 en admet 2.

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0019

```
def nbdiviseur(n):
    c=1
    for d in range(2,n+1):
        if diviseur(d,n)==True:
            c=c+1
    return c
```

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> nbdiviseur(10)
4
>>> nbdiviseur(30)
8
>>> nbdiviseur(31)
2
```

go.eyrolles.com/ti-python7

Exercice 1

Écrire une fonction `pgrandiv` qui prend comme argument un entier supérieur strict à 1 et qui renvoie le plus grand diviseur de `n` inférieur strict à *n*.

Corrigé de l'exercice page xxx

Objectif 3 Nous allons construire une fonction `listediv(n)` avec un paramètre *n* entier naturel supérieur strict à 1 et qui renvoie tous les diviseurs positifs de *n*.

On va utiliser une liste `l` dans laquelle seront stockés les diviseurs de *n* que l'on va rencontrer au fur et à mesure. Pour indiquer que `l` est une liste, il faut l'initialiser comme une liste vide en écrivant `l=[ ]`.

Algorithme

```
fonction listediv(n):
   l←[ ]
      Pour d allant de 1 à n
         Si diviseur(d,n)=True alors
            Ajouter à la liste l la
valeur d
         FinSi
      FinPour
   renvoyer l
Fin
```

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0027

```
def listediv(n):
    l=[]
    for d in range(1,n+1,):
        if diviseur(d,n)==True:
            l.append(d)
    return l
```

Déterminons les diviseurs de 10 : on trouve 1 , 2, 5 et 10.

Les diviseurs de 30 sont 1, 2, 3, 5, 6, 10, 15 et 30 ; ceux de 31 sont 1 et 31.

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> listediv(10)
[1, 2, 5, 10]
>>> listediv(30)
[1, 2, 3, 5, 6, 10, 15, 30]
>>> listediv(31)
[1, 31]
```

Exercice 2

Écrire une fonction `sommediv` qui prend comme argument l'entier n et qui renvoie la somme des diviseurs de n.

Corrigé de l'exercice page xxx

Théorème : Soit n un entier naturel non nul. n est un nombre premier dès que n admet exactement deux diviseurs : 1 et n.

Objectif 4 Nous allons construire une fonction `premier1` avec un paramètre n entier naturel supérieur strict à 1 et qui renvoie `True` si n est premier, `False` sinon.

On utilisera la fonction `nbdiviseur` précédente.

Algorithme

```
fonction premier1(n):
   k←nbdiviseur(n)
   Si k=2 alors
      Renvoyer True
   Sinon
      Renvoyer False
   FinSi
Fin
```

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0035

```
def premier1(n):
    k=nbdiviseur(n)
    if k==2:
        return True
    else:
        return False
```

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> premier1(2)
True
>>> premier1(18)
False
>>> premier1(19)
True
```

1 et 18 ne sont pas premiers alors que 2 l'est.

Théorème : n n'est pas premier si et seulement si il admet un diviseur d vérifiant $1 < d < \sqrt{n}$.

Objectif 5 Nous allons construire une fonction `premier2` avec un paramètre n entier naturel supérieur strict à 1 et qui renvoie 1 si n est premier, d un diviseur de n avec $1 < d < n$ si n n'est pas premier.

On va donc calculer $m = E\left(\sqrt{n}\right)$, puis tester si i est un diviseur de n pour i allant de 2 à m. Si on trouve un diviseur, il faudra le stocker dans une variable k (qu'on va initialiser à 1 et qui restera égale à 1 si on ne trouve pas de diviseur). On renverra k à la fin.

On rappelle que le théorème, et donc l'algorithme, suppose que $n > 1$.

Enfin, pour utiliser la fonction racine carrée, il faudra importer la bibliothèque math.

Algorithme

```
fonction premier2(n):
   m ← E(√n)
   k ← 1
   Pour i allant de 2 à m
      Si i divise n alors
         k ← i
   FinPour
   Renvoyer k
Fin
```

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0042

```
from math import *
def premier2(n):
    m=floor(sqrt(n))
    k=1
    for i in range(2,m+1):
        if n%i==0:
            k=i
    return k
```

Exécutons maintenant cette fonction en console.

2 est premier, alors la fonction renvoie bien 1.

30 n'est pas premier, alors la fonction renvoie 5 (le dernier diviseur trouvé dans la boucle).

17 est premier, alors la fonction renvoie 1.

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> premier2(2)
1
>>> premier2(30)
5
>>> premier2(17)
1
```

go.eyrolles.com/ti-python8

Objectif 6 Nous allons construire une fonction `supremier` avec un paramètre *n* entier naturel supérieur ou égal à 1 et qui renvoie le plus petit nombre premier supérieur ou égal à *n*.

Nous allons écrire une boucle `TantQue` avec une variable *i* qui commencera à *n* puis nous allons tester si *i* est premier ou non. Nous nous arrêterons lorsqu'on aura trouvé un nombre premier, sinon nous incrémenterons *i* de 1.

Algorithme

```
fonction supremier(n):
   i ← n
   TantQue premier1(i)=False
      i ← i+1
   FinTantQue
   Renvoyer i
Fin
```

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0051

```
def supremier(n):
    i=n
    while premier1(i)==False:
        i=i+1
    return i
```

Exécutons maintenant cette fonction en console.

2 est premier, alors la fonction renvoie bien 1.

30 n'est pas premier, alors la fonction renvoie 5 (le dernier diviseur trouvé dans la boucle).

17 est premier, alors la fonction renvoie 1.

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> premier2(2)
1
>>> premier2(30)
5
>>> premier2(17)
1
```

Exercice 3

Écrire une fonction `infpremier` qui prend comme argument un entier naturel *n* supérieur strict à 1 et qui renvoie le plus grand nombre premier inférieur ou égal à *n*.

Corrigé de l'exercice page xxx

Objectif 7 Nous allons à présent écrire une fonction `nieme` qui prend comme argument un entier naturel n supérieur ou égal à 1 et qui renvoie le n-ième nombre premier.

Algorithme

```
Fonction nieme(n)
   i←0
   j←2
   TantQue i<n
      Si premier1(j)=1 alors
         i←i+1
      FinSi
      j←j+1
   Fin TantQue
   Renvoyer j-1
Fin
```

On vérifie bien que notre script fonctionne en l'exécutant.

Les nombres premiers sont : 2, 3, 5, 7, 11… donc le 5e nombre premier est bien 11.

Le 40e nombre premier est 173.

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0057

```
def nieme(n):
    i=0
    j=2
    while i<n:
        if premier1(j)==1:
            i=i+1
        j=j+1
    return j-1
```

On renvoie $j-1$ car j est incrémenté de 1 en fin de boucle.

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> nieme(5)
11
>>> nieme(40)
173
```

Objectif 8 Nous allons écrire une fonction `ninfprems` qui prend comme argument un entier naturel n supérieur strict à 1 et qui renvoie la liste des nombres premiers inférieurs ou égaux à n.

Algorithme

```
fonction ninfprems(n)
   l←[]
   Pour i allant de 1 à n
      Si premier1(i)=True alors
         Ajouter i à la liste l
      FinSi
   Fin Pour
   Renvoyer l
Fin
```

Les nombres premiers inférieurs à 50 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 et 47.

Script en Python :

ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0066

```
def ninfprems(n):
    l=[]
    for i in range(1,n+1):
        if premier1(i)==True:
            l.append(i)
    return l
```

PYTHON SHELL

```
>>> # L'exécution de PREMIER
>>> from PREMIER import *
>>> ninfprems(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23,
 29, 31, 37, 41, 43, 47]
```

Exercice 4

Écrire une fonction `npremier` qui prend comme argument un entier naturel n supérieur ou égal à 1 et qui renvoie la liste des n premiers nombres premiers.

Corrigé de l'exercice page xxx