Automates et Récursivité

Daniel Porumbel
dp.cnam@gmail.com

nombreaux slides dus à
Pierre Cubaud

Plan

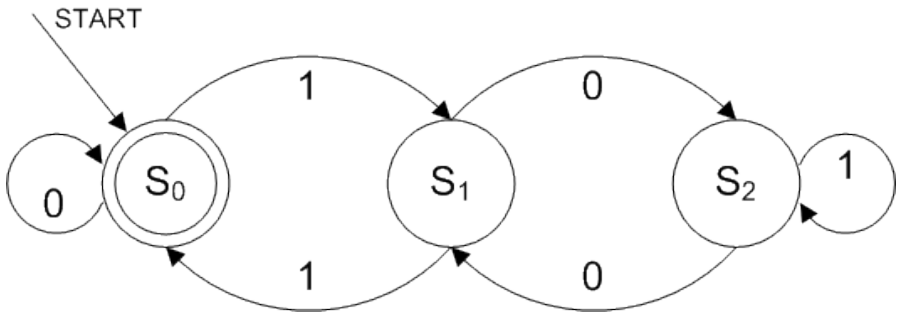
- Les notions de base d'automate
- Les fonctions récursives

1) les automates à états finis

- défini par un ensemble fini d'états
- le changement d'état se produit par une transition liée à l'occurence d'un "évenement"

- une action peut se produire dans un état (automate de Moore) ou lors d'une transition (automate de Mealey)

- représentation graphique et/ou table de transition



(wikipedia)

Table de transition

	0	1
s_0	s_0	s_1
s_1	s_2	s_0
s_2	s_1	s_2

L'état d'une case à cocher 1

Rappel code : Quelle est la signification de la variable `state` ?

```
int state = 0;
void draw() {
  background(200);
  stroke(0);noFill();
  if ((state==1)||(state==3))
    strokeWeight(4);
  else
    strokeWeight(1);
  rect(50,50,40,40);
  if ((state==2)||(state==3)) {//States 2,3 = ?
    noStroke();fill(255,0,0);
    rect(50+10,50+10,20,20);
  }
}
```

L'état d'une case à cocher 2

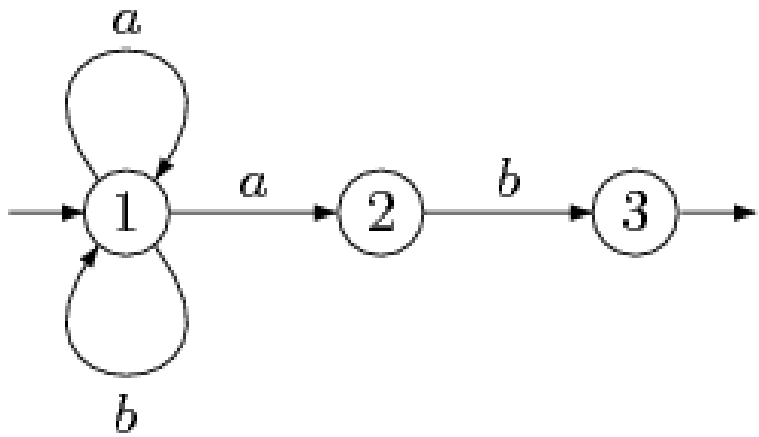
Rappel code : Quelle est la signification de la variable `state` ?

```
void mouseMoved() {
 int x = mouseX;
 int y = mouseY;
 if((x > 50)&&(x<50+40)&&(y>50)&&(y<50+40)){
        if(state==0) state = 1;
        if(state==2) state = 3;
 } else{
    if(state==1) state = 0;
    if(state==3) state = 2;
 }
}
void mousePressed() {
 if (state==1)state = 3 ;
 else if (state==3) state = 1 ;
}
```

Application à l'analyse lexicale

Dans ce cas, les transitions sont les caractères du mot à analyser

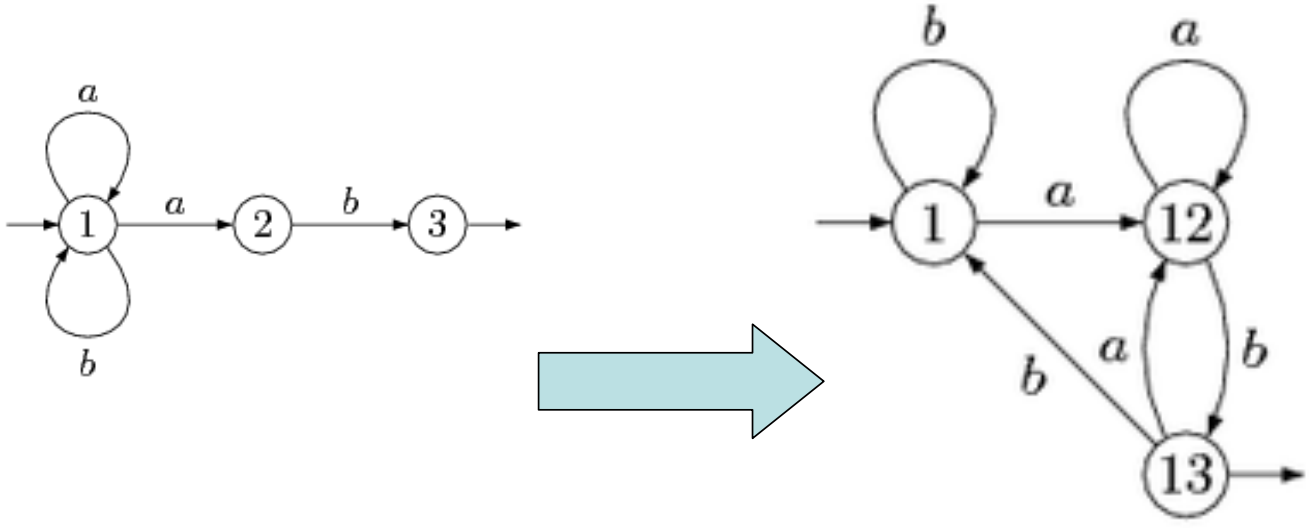
Exemple : reconnaître les mots qui terminent par ab



On a ici un automate fini non déterministe (AFN) : une case de la table des états peut mener vers deux (ou plus) états différents

AFD = Automate fini déterministe

On peut transformer tout AFN en un AFD !



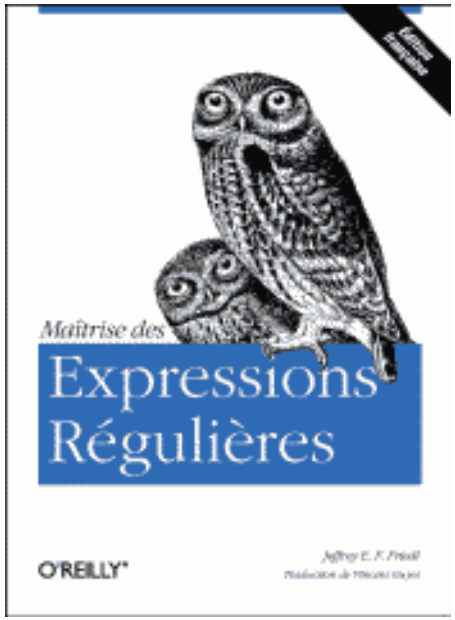
Mais cela peut générer une explosion du nombre d'états

Expressions régulières

utilisées par grep, sed, awk, perl, java, etc.

[abc...]
matches any of the characters *abc*...
[^abc...]
matches any character except *abc*...
r1|r2
matches either *r1* or *r2* (alternation).
r1r2
matches *r1*, and then *r2* (concatenation).
r+
matches one or more *r*'s.
r*
matches zero or more *r*'s.
r?
matches zero or one *r*'s.
(r)
matches *r* (grouping).
r{n,m}
matches at least *n*, *n* to any number, or *n* to *m* occurrences of *r* (interval expressions).
\b
matches the empty string at either the beginning or the end of a word.

etc ...



Rappel : Fonctions

Lorsqu'on appelle une fonction, il y a trois étapes.

L'entrée on fait « rentrer » des informations dans la fonction (on lui donne les données avec lesquelles travailler).

Les calculs grâce aux informations qu'elle a reçues en entrée, la fonction travaille.

La sortie une fois qu'elle a fini ses calculs, la fonction renvoie un résultat.

Quel est le résultat du code ?

```
int cube(int x){
    int y = x * x * x;
    return y;
}
void setup(){
    int a = 3;
    int b = cube(a);
    println(b);
}
```

Fonctions récursives

- une fonction récursive est une fonction qui s'appelle elle-même
- Quel est le résultat du code ?

```
int f(int n){
  if(n==1)
        return 1;
  else
        return f(n-1)+n;
}
void setup(){
    int s = f(3);
    println(s);
}
```

2) La récursivité

Exemple du calcul de la factorielle (cf ED3)

$$n! = n.(n-1).(n-2)...2.1$$

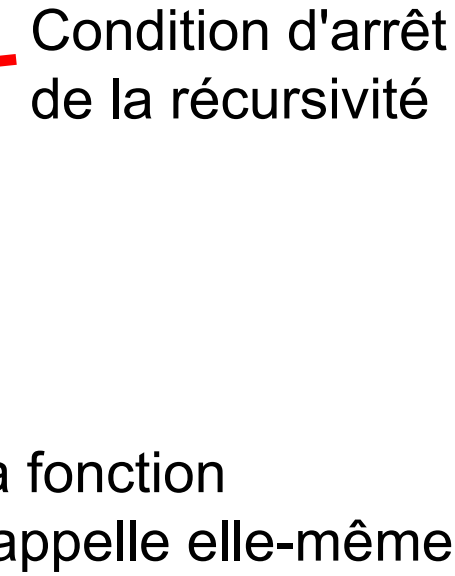
factorielle

```
int N=25;
int i=1;
float P=1;

while (i<N) {
  i = i + 1;
  P = P * i;
}

println(i+"    "+P);
```

On peut exploiter
n! = n*(n-1)!

factorielleREC

```
void setup(){
  int n = factorielle(25);
  println(n);
}

int factorielle(int n) {
  if (n == 1)
    return 1;
  else
    return n*factorielle(n-1);
}
```



Autre exemple : la suite de Fibonacci (ED3 encore)

$$F_1 = F_2 = 1 \ , F_{n+2} = F_n + F_{n+1} \ , n \geq 0$$

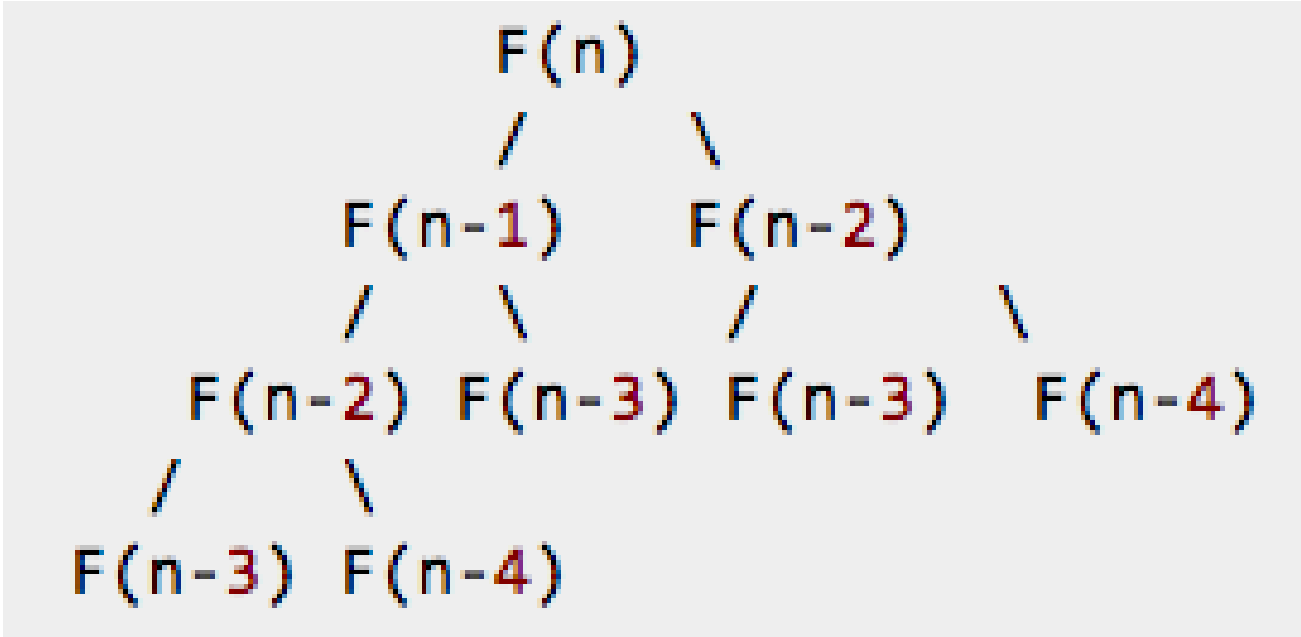
fibonacciREC

```
void setup(){
  int n = fibonacci(10);
  println(n);
}

int fibonacci(int n) {
  if (n <= 2)
    return 1;
  else
    return fibonacci(n-1)+fibonacci(n-2);
}
```

```
Display 0 does not exist, using the default display instead.
55
```

Rque : si n > 48 on dépasse le max int (passer en long ou float)

Appels successifs à la fonction :

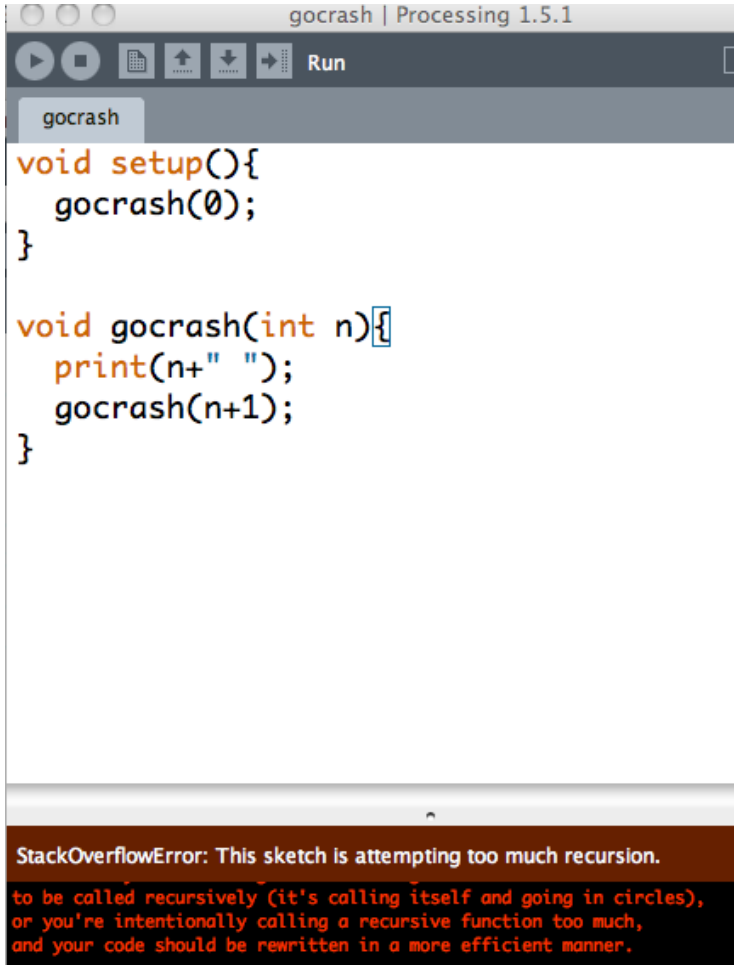


On fait 2^n appels à la fonction : c'est lent !!

Il y a aussi plein de calculs inutiles

Les paramètres d'appel de la fonction sont stockés dans une mémoire spécifique : une pile

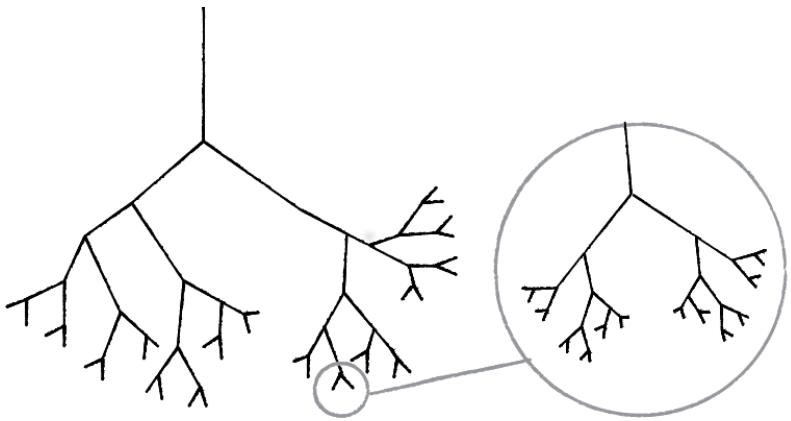
Celle-ci est forcément limitée en taille



En programmation récursive, il faut savoir maitriser la pile des appels !

Figures fractales

- un fractale se compose de figures qui contiennent elles-même une figure identique
  - le fractale se contient lui-même
  - il est recopie à plusieurs échelles (de zoom)

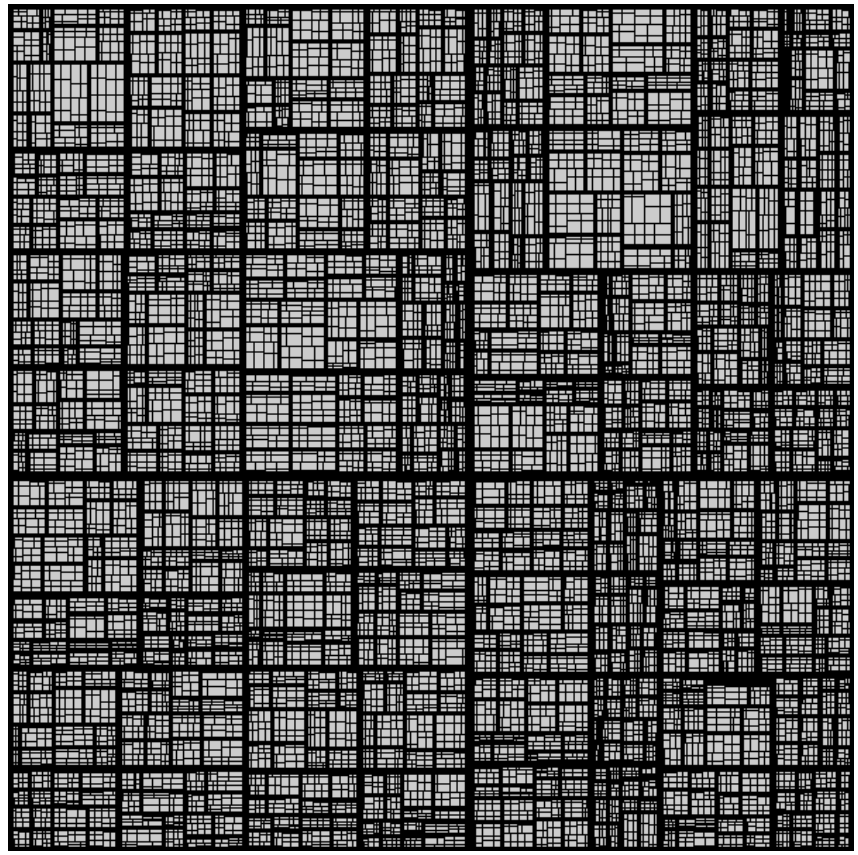


- cet arbre a été crée par un programme récursif de < 10 lignes

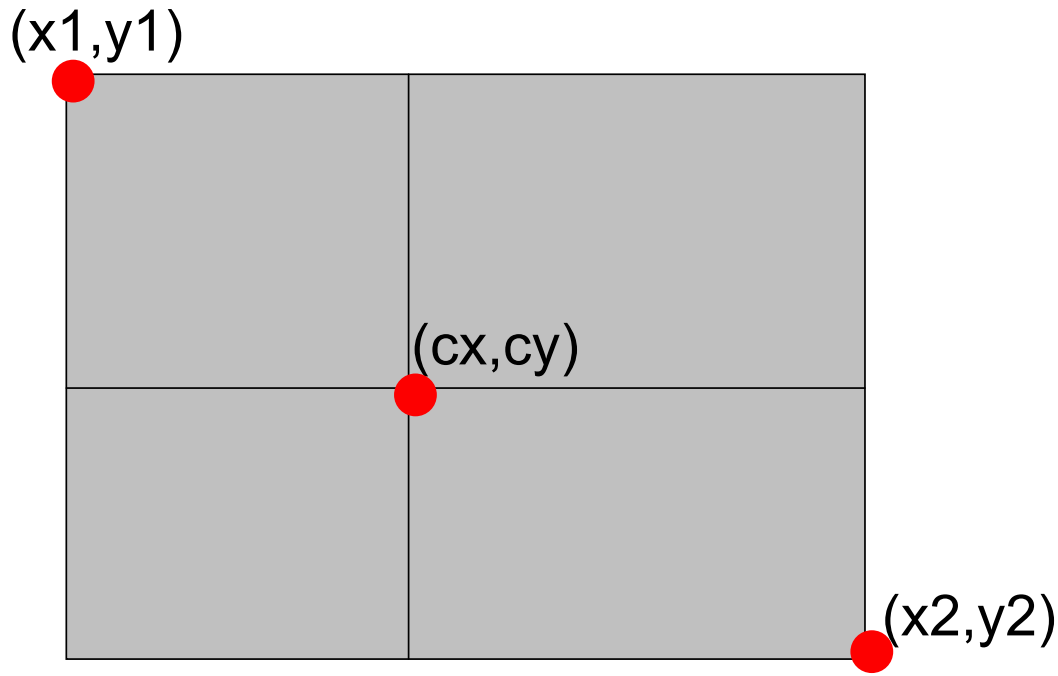
Exemple de code Image Fractale

```
void fractale(float x,float y
                        ,float dst, float ang){
    float xFin=  x+dst*sin(ang);
    float yFin = y+dst*cos(ang);
    if(dst>10){
        line(x,y,xFin,yFin);
        fractale(xFin,yFin, dst*0.66, ang+1);
        fractale(xFin,yFin, dst*0.66, ang-1);
    }
}
void setup(){
    size(600,500);
    fractale(300.0,100.0, 100.0, 0.0);
};
```

Autre exemple : subdivision récursive du plan



On part d'un rectangle défini par ses coins (x1,y1) et (x2,y2) :



On tire au sort un point à l'intérieur (cx,cy)

puis on subdivide les 4 rectangles obtenus

subdiv

```
void setup(){
  size(600,600);
  rectMode(CORNERS);
  subdiv(20,0,0,width,height);
}

void subdiv(int n,int x1,int y1,int x2,int y2) {
 rect(x1,y1,x2,y2);
 if (n>0) {
  int cx = (x1+x2)/2; // + du random
  int cy = (y1+y2)/2; // + du random
  subdiv(n-1,x1,y1,cx,cy);
  subdiv(n-1,cx,y1,x2,cy);
  subdiv(n-1,x1,cy,cx,y2);
  subdiv(n-1,cx,cy,x2,y2);
 }
}
```

Soit le code :

```
int f(int n){
    if (n<=2){
        return 1;
    }else{
        return f(n-1)*f(n-1)+f(n-2)*f(n-2)
    }
}
```

- quelle est la valeur de $f(5)$?
- peut-on réduire le nombre de calculs ?

Quel est le résultat du code ?

```
void rec(int n, int[] tab){
    if (n<tab.length){
        rec(n+1,tab);
        println(tab[n]);
    }
}
void setup(){
    int[] tab = {1, 2, 5, 9};
    rec(0,tab);
}
```

Quelques exercices

1. Écrire une fonction récursive pour calculer

$$2 + 4 + 6 + \cdots + n,$$

où n est un nombre pair

Quelques exercices

1. Écrire une fonction récursive pour calculer

$$2 + 4 + 6 + \cdots + n,$$

où n est un nombre pair

2. Écrire une fonction récursive qui calcule la somme des n premiers carrés : $1^2 + 2^2 + 3^2 + \ldots n^2$

Quelques exercices

1. Écrire une fonction récursive pour calculer

$$2 + 4 + 6 + \cdots + n,$$

où n est un nombre pair

2. Écrire une fonction récursive qui calcule la somme des n premiers carrés : $1^2 + 2^2 + 3^2 + \ldots n^2$
3. Écrire une fonction récursive qui calcule la somme des éléments positifs d'un tableau

Quelques exercices

1. Écrire une fonction récursive pour calculer

$$2 + 4 + 6 + \cdots + n,$$

où n est un nombre pair

2. Écrire une fonction récursive qui calcule la somme des n premiers carrés : $1^2 + 2^2 + 3^2 + \ldots n^2$
3. Écrire une fonction récursive qui calcule la somme des éléments positifs d'un tableau
4. Écrire une fonction récursive pour calculer la fonction suivante :

$$F_1 = F_2 = F_3 = 1$$

$$F_n = F_{n-1} + F_{n-2} + F_{n-3}, \forall n \geq 4$$

Exercices 2

Appliquer l'algorithme d'Euclid pour calculer le `pgcd` (plus grand commun diviseur) de a et b.

- si $a > b$, on a $\texttt{pgcg}(a, b) = \texttt{pgcd}(a - b, b)$
- si $a < b$, on a $\texttt{pgcg}(a, b) = \texttt{pgcd}(a, b - a)$
- si $a = b$, on a $\texttt{pgcg}(a, b) = a$

Calculer cela à l'aide d'une fonction récursive.

Exercices 3

écrire une fonction récursive pour calculer la fonction d'Ackermann :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ et } n > 0. \end{cases}$$