Mathématiques discrètes IV
Arithmétique
INFO1 - Semaine 4

Guillaume CONNAN

IUT de Nantes - Dpt d'informatique

Dernière mise à jour : 18 février 2013

Sommaire

Sommaire

Sommaire

Définition 1 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

Définition 1 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{R}, =)$

Définition 1 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 1 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 1 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 2 (Élément neutre)

C'est un élément de E qui vérifie, pour tout élément x de E,

$$e \star x = x \star e = x$$

Définition 2 (Élément neutre)

C'est un élément de E qui vérifie, pour tout élément x de E,

$$e \star x = x \star e = x$$

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 2 (Élément neutre)

C'est un élément de E qui vérifie, pour tout élément x de E,

$$e \star x = x \star e = x$$

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 2 (Élément neutre)

C'est un élément de E qui vérifie, pour tout élément x de E,

$$e \star x = x \star e = x$$

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 2 (Élément neutre)

C'est un élément de E qui vérifie, pour tout élément x de E,

$$e \star x = x \star e = x$$

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 3 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

Définition 3 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{R}, =)$

Définition 3 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 3 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Définition 3 (Monoïde)

On appelle monoïde tout couple $(E, \star)$ tel que la loi $\star$ soit *associative*. Si en plus le monoïde admet un élément neutre, on dit que le monoïde est **libre** (ou **unitaire**).

- $(\mathbb{N}, +)$
- $(\mathcal{P}(E), \cap)$
- $(\mathbb{N}, -)$

Théorème 4 (Sous-monoïde)

Soit $(E, \star)$ un monoïde et $M \subseteq E$ tel que M soit stable par $\star$. Alors $(M, \star)$ est un monoïde : c'est un ***sous-monoïde*** *de $(E, \star)$.*

Définition 5 (Élément inversible)

Soit $(E, \star)$ un monoïde unitaire d'élément neutre e. Un élément x de E est inversible (ou symétrisable) par rapport à $\star$ s'il existe y dans E tel que

$$x \star y = y \star x = e$$

Sommaire

Définition 6 (Groupe)

Un groupe est un monoïde unitaire tel que tout élément est inversible.

Définition 7 (Groupe fini)

Soit $(G, +)$ un groupe d'élément neutre 0_G ayant un nombre fini d'éléments.
Le cardinal de G est appelé l'ordre du groupe G.
Soit x un element de G. L'ordre de l'élément x est le plus petit entier k tel que $k \cdot x = 0$.

Définition 6 (Groupe)

Un groupe est un monoïde unitaire tel que tout élément est inversible.

Définition 7 (Groupe fini)

Soit $(G, +)$ un groupe d'élément neutre 0_G ayant un nombre fini d'éléments.
Le cardinal de G est appelé l'ordre du groupe G.
Soit x un élément de G. L'ordre de l'élément x est le plus petit entier k tel que $k \cdot x = 0$.

Définition 6 (Groupe)

Un groupe est un monoïde unitaire tel que tout élément est inversible.

Définition 7 (Groupe fini)

Soit $(G,+)$ un groupe d'élément neutre 0_E ayant un nombre fini d'éléments.
Le cardinal de G est appelé l'**ordre du groupe** G.
Soit x un élément de G. L'**ordre de l'élément** x est le plus petit entier k tel que $k \cdot x = 0$.

Sommaire

Théorème 8 (Division euclidienne)

Soit a un entier relatif et b un entier naturel non nul.
Il existe un unique couple d'entiers (q, r) tels que

$$a = bq + r \quad \text{avec} \quad 0 \leqslant r < b$$

Déterminer q et r, c'est effectuer la division euclidienne de a par b.
On appelle a le dividende, b le diviseur, q le quotient et r le reste.

recherche

Pourquoi dit-on habituellement qu'on ne peut pas diviser par zéro ?
D'ailleurs, dans de nombreux cas, le debugger de votre langage préféré renvoie des messages du type `Exception: Division_by_zero`...

remarque

Si l'on remplace la condition sur le reste par $0 \leqslant |r| < b$, il n'y a plus unicité (c'est d'ailleurs la formulation habituelle qui permet de travailler dans les mêmes conditions dans des structures plus vastes qu'on appelle *anneaux euclidiens*) ce qui explique que Python et Ocaml ne sont pas d'accord...

Recherche

Que devient le théorème de la division euclidienne dans le cas où $a < 0$?

remarque

Si l'on remplace la condition sur le reste par $0 \leqslant |r| < b$, il n'y a plus unicité (c'est d'ailleurs la formulation habituelle qui permet de travailler dans les mêmes conditions dans des structures plus vastes qu'on appelle *anneaux euclidiens*) ce qui explique que Python et Ocaml ne sont pas d'accord...

recherche

Que devient le théorème de la division euclidienne dans le cas où $a \leqslant 0$?

remarque

Si l'on remplace la condition sur le reste par $0 \leqslant |r| < b$, il n'y a plus unicité (c'est d'ailleurs la formulation habituelle qui permet de travailler dans les mêmes conditions dans des structures plus vastes qu'on appelle *anneaux euclidiens*) ce qui explique que Python et Ocaml ne sont pas d'accord...

recherche

Que devient le théorève de la division euclidienne dans le cas où $a < 0$?

Sommaire

Sommaire

Définition 9 (Congruence des entiers)

Soit n un entier naturel non nul.
Deux éléments a et b du groupe $(\mathbb{Z},+)$ sont **congrus modulo** n si, et seulement si, ils ont le même reste dans la division par n. On note

$$a \equiv b \ (\mathrm{mod}\, n)$$

et on lit « *a est congru à b modulo n* ».

« *Well, LISP seems to work okay now, modulo that GC bug.* »

Définition 9 (Congruence des entiers)

Soit n un entier naturel non nul.
Deux éléments a et b du groupe $(\mathbb{Z},+)$ sont **congrus modulo** n si, et seulement si, ils ont le même reste dans la division par n. On note

$$a \equiv b \ (\mathrm{mod}\, n)$$

et on lit « *a est congru à b modulo n* ».

« Well, LISP seems to work okay now, modulo that GC bug. »
« I feel fine today, modulo a slight headache. »

Définition 9 (Congruence des entiers)

Soit n un entier naturel non nul.
Deux éléments a et b du groupe $(\mathbb{Z},+)$ sont **congrus modulo** n si, et seulement si, ils ont le même reste dans la division par n. On note

$$a \equiv b \ (\mathrm{mod}\, n)$$

et on lit « *a est congru à b modulo n* ».

« Well, LISP seems to work okay now, modulo that GC bug. »
« I feel fine today modulo a slight headache. »

in « The New Hacker's Dictionnary »
http ://outpost9.com/reference/jargon/jargon_28.html#SEC35

Définition 9 (Congruence des entiers)

Soit n un entier naturel non nul.
Deux éléments a et b du groupe $(\mathbb{Z},+)$ sont **congrus modulo** n si, et seulement si, ils ont le même reste dans la division par n. On note

$$a \equiv b \pmod{n}$$

et on lit « *a est congru à b modulo n* ».

« Well, LISP seems to work okay now, modulo that GC bug. »
« I feel fine today modulo a slight headache. »

in « The New Hacker's Dictionnary »
http ://outpost9.com/reference/jargon/jargon_28.html#SEC35

Définition 9 (Congruence des entiers)

Soit n un entier naturel non nul.
Deux éléments a et b du groupe $(\mathbb{Z},+)$ sont **congrus modulo** n si, et seulement si, ils ont le même reste dans la division par n. On note

$$a \equiv b \pmod{n}$$

et on lit « *a est congru à b modulo n* ».

« Well, LISP seems to work okay now, modulo that GC bug. »
« I feel fine today modulo a slight headache. »

in « The New Hacker's Dictionnary »
http ://outpost9.com/reference/jargon/jargon_28.html#SEC35

Définition 9 (Congruence des entiers)

Soit n un entier naturel non nul.
Deux éléments a et b du groupe $(\mathbb{Z},+)$ sont **congrus modulo** n si, et seulement si, ils ont le même reste dans la division par n. On note

$$a \equiv b \pmod{n}$$

et on lit « *a est congru à b modulo n* ».

« Well, LISP seems to work okay now, modulo that GC bug. »
« I feel fine today modulo a slight headache. »

in « The New Hacker's Dictionnary »
http ://outpost9.com/reference/jargon/jargon_28.html#SEC35

```
        Objective Caml version 3.12.0

# 17 mod 3 ;;
- : int = 2
# -17 mod 3 ;;
- : int = -2
# -17/3;;
- : int = -5
# (-17 / 3)*3 + (-17 mod 3) ;;
- : int = -17
```

```
Python 2.7.2+ (default, Oct 4 2011, 20:06:09)
>>> 17 % 3
2
>>> -17 % 3
1
>>> -17/3
-6
>>> (-17 // 3)*3 + (-17 % 3)
-17
```

```
        Objective Caml version 3.12.0

# 17 mod 3 ;;
- : int = 2
# -17 mod 3 ;;
- : int = -2
# -17/3;;
- : int = -5
# (-17 / 3)*3 + (-17 mod 3) ;;
- : int = -17
```

```
Python 2.7.2+ (default, Oct 4 2011, 20:06:09)
>>> 17 % 3
2
>>> -17 % 3
1
>>> -17/3
-6
>>> (-17 // 3)*3 + (-17 % 3)
-17
```

En général, on prend comme *représentant principal* de la classe de a modulo n le reste de la division de a par n. Par exemple, on notera :

$$\mathbb{Z}/8\mathbb{Z} = \{\overline{0}^8, \overline{1}^8, \overline{2}^8, \overline{3}^8, \overline{4}^8, \overline{5}^8, \overline{6}^8, \overline{7}^8\}$$

DANGER !

Notez bien que $\overline{0}^8$, $\overline{1}^8$, etc. sont des ensembles d'entiers !...

Ici, $\overline{0}^8 = \{..., -16, -8, 0, 8, 16, 24, 32, ...\}$

Par **ABUS DE NOTATION**, on écrira le plus souvent k pour désigner $\overline{k}^n$ ce qui nous amènera à écrire par exemple que $8 = 0$, voire $3 \times 5 = 7 = -1$ comme nous le verrons plus loin...

DANGER !

Notez bien que $\overline{0}^8$, $\overline{1}^8$, etc. sont des ensembles d'entiers !...

Ici, $\overline{0}^8 = \{..., -16, -8, 0, 8, 16, 24, 32, ...\}$

Par **ABUS DE NOTATION**, on écrira le plus souvent k pour désigner $\overline{k}^n$ ce qui nous amènera à écrire par exemple que $8 = 0$, voire $3 \times 5 = 7 = -1$ comme nous le verrons plus loin...

DANGER !

Notez bien que $\overline{0}^8$, $\overline{1}^8$, etc. sont des ensembles d'entiers !...

Ici, $\overline{0}^8 = \{..., -16, -8, 0, 8, 16, 24, 32, ...\}$

Par **ABUS DE NOTATION**, on écrira le plus souvent k pour désigner $\overline{k}^n$ ce qui nous amènera à écrire par exemple que $8 = 0$, voire $3 \times 5 = 7 = -1$ comme nous le verrons plus loin...

Théorème 10

$$a \equiv b (\mathrm{mod}\, n) \Longleftrightarrow \exists k \in \mathbb{Z} \mid a = b + kn$$

Théorème 11

1. $\text{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad ; \quad a - a' \equiv b - b' \pmod{n} \quad ; \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Théorème 11

1. $\text{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad ; \quad a - a' \equiv b - b' \pmod{n} \quad ; \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Théorème 11

1. $\text{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad a - a' \equiv b - b' \pmod{n} \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Théorème 11

1. $\mathrm{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad a - a' \equiv b - b' \pmod{n} \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Théorème 11

1. $\text{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad a - a' \equiv b - b' \pmod{n} \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Théorème 11

1. $\text{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad a - a' \equiv b - b' \pmod{n} \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Théorème 11

1. $\text{Card}\left(\mathbb{Z}/_{n\mathbb{Z}}\right) = n$
2. $a \equiv a \pmod{n}$
3. *Si* $a \equiv b \pmod{n}$, *alors* $b \equiv a \pmod{n}$
4. *Si* $a \equiv b \pmod{n}$ *et* $b \equiv c \pmod{n}$, *alors* $a \equiv c \pmod{n}$
5. *Si* $a \equiv b \pmod{n}$ *et* $a' \equiv b' \pmod{n}$, *alors*

$$a + a' \equiv b + b' \pmod{n} \quad a - a' \equiv b - b' \pmod{n} \quad aa' \equiv bb' \pmod{n}$$

6. *Si* $a \equiv b \pmod{n}$, *alors, pour tout* $p \in \mathbb{N}$, $a^p \equiv b^p \pmod{n}$

Attention !

$12 \equiv 0 (\mathrm{mod}\, 6)$ $\frac{12}{3} \not\equiv \frac{0}{3} (\mathrm{mod}\, 6)$.

Attention !

$12 \equiv 0 (\mathrm{mod}\, 6)$ $\frac{12}{3} \not\equiv \frac{0}{3} (\mathrm{mod}\, 6)$.

Attention !

$12 \equiv 0 (\mathrm{mod}\, 6)$ $\frac{12}{3} \not\equiv \frac{0}{3} (\mathrm{mod}\, 6)$.

Théorème 12

$$\forall n \in \mathbb{N}^*,\ \forall (x,y) \in \mathbb{Z}^2,\quad \overline{x}^n + \overline{y}^n = \overline{x+y}^n,\quad \overline{x}^n \cdot \overline{y}^n = \overline{x \cdot y}^n$$

Écrivez par exemple les lois d'addition et de multiplication de $\mathbb{Z}/7\mathbb{Z}$ et $\mathbb{Z}/8\mathbb{Z}$: des remarques ?
Est-ce que $(\mathbb{Z}/n\mathbb{Z}, +)$ et $(\mathbb{Z}/n\mathbb{Z}, \cdot)$ sont des groupes ?

Théorème 12

$$\forall n \in \mathbb{N}^*,\ \forall (x,y) \in \mathbb{Z}^2,\quad \overline{x}^n + \overline{y}^n = \overline{x+y}^n,\quad \overline{x}^n \cdot \overline{y}^n = \overline{x \cdot y}^n$$

Écrivez par exemple les lois d'addition et de multiplication de $\mathbb{Z}/7\mathbb{Z}$ et $\mathbb{Z}/8\mathbb{Z}$: des remarques ?
Est-ce que $(\mathbb{Z}/n\mathbb{Z},+)$ et $(\mathbb{Z}/n\mathbb{Z},\cdot)$ sont des groupes ?

Théorème 12

$\forall n \in \mathbb{N}^*,\ \forall (x,y) \in \mathbb{Z}^2,\quad \overline{x}^n + \overline{y}^n = \overline{x+y}^n,\quad \overline{x}^n \cdot \overline{y}^n = \overline{x \cdot y}^n$

Écrivez par exemple les lois d'addition et de multiplication de $\mathbb{Z}/7\mathbb{Z}$ et $\mathbb{Z}/8\mathbb{Z}$: des remarques ?

Est-ce que $(\mathbb{Z}/n\mathbb{Z}, +)$ et $(\mathbb{Z}/n\mathbb{Z}, \cdot)$ sont des groupes ?

Théorème 12

$\forall n \in \mathbb{N}^*,\ \forall (x,y) \in \mathbb{Z}^2,\quad \overline{x}^n + \overline{y}^n = \overline{x+y}^n,\quad \overline{x}^n \cdot \overline{y}^n = \overline{x \cdot y}^n$

Écrivez par exemple les lois d'addition et de multiplication de $\mathbb{Z}/7\mathbb{Z}$ et $\mathbb{Z}/8\mathbb{Z}$: des remarques ?
Est-ce que $(\mathbb{Z}/n\mathbb{Z},+)$ et $(\mathbb{Z}/n\mathbb{Z},\cdot)$ sont des groupes ?

Sommaire

```
type classe  =
  {representant : int ; modulo : int};;
```

```
let (%) a n = {representant = (a mod n); modulo = n};;
```

```
# 13 % 3;;
- : classe = {representant = 1; modulo = 3}
```

```
type classe  =
  {representant : int ; modulo : int};;
```

```
let (%) a n = {representant = (a mod n); modulo = n};;
```

```
# 13 % 3;;
- : classe = {representant = 1; modulo = 3}
```

```
type classe  =
  {representant : int ; modulo : int};;
```

```
let (%) a n = {representant = (a mod n); modulo = n};;
```

```
# 13 % 3;;
- : classe = {representant = 1; modulo = 3}
```

```
exception Classes_incompatibles;;

let loi_induite loi =  fun a  b ->
  {modulo =
      if a.modulo = b.modulo
      then a.modulo
      else raise Classes_incompatibles ;
   representant =
      (loi (a.representant) (b.representant)) mod (a.modulo)
  };;

let ( +% ) = loi_induite  ( + ) ;;
let ( *% ) = loi_induite  ( * ) ;;
```

```
# (9%5) *% (7%5);;
- : classe = {representant = 3; modulo = 5}
```

```
exception Classes_incompatibles;;

let loi_induite loi =  fun a  b ->
  {modulo =
      if a.modulo = b.modulo
      then a.modulo
      else raise Classes_incompatibles ;
   representant =
      (loi (a.representant) (b.representant)) mod (a.modulo)
  };;

let ( +% ) = loi_induite  ( + ) ;;
let ( *% ) = loi_induite  ( * ) ;;
```

```
# (9%5) *% (7%5);;
- : classe = {representant = 3; modulo = 5}
```

Sommaire

Théorème 13

Toute partie non vide de $\mathbb{N}$ possède un plus petit élément.

Théorème 14

Toute partie non vide MAJOREE de $\mathbb{N}$ possède un plus GRAND élément.

Théorème 13

Toute partie non vide de $\mathbb{N}$ possède un plus petit élément.

Théorème 14

Toute partie non vide MAJORÉE de $\mathbb{N}$ possède un plus GRAND élément.

Théorème 13

Toute partie non vide de $\mathbb{N}$ possède un plus petit élément.

Théorème 14

Toute partie non vide MAJORÉE de $\mathbb{N}$ possède un plus GRAND élément.

Théorème 15

Si d divise a et b alors d divise toute COMBINAISON LINEAIRE de a et b

Théorème 13

Toute partie non vide de $\mathbb{N}$ possède un plus petit élément.

Théorème 14

Toute partie non vide MAJORÉE de $\mathbb{N}$ possède un plus GRAND élément.

Théorème 15

Si d divise a et b alors d divise toute COMBINAISON LINÉAIRE de a et b.

Théorème 13

Toute partie non vide de $\mathbb{N}$ possède un plus petit élément.

Théorème 14

Toute partie non vide MAJORÉE de $\mathbb{N}$ possède un plus GRAND élément.

Théorème 15

Si d divise a et b alors d divise toute COMBINAISON LINÉAIRE de a et b.

Théorème 16

Si d divise a alors $|d| \leqslant |a|$.

Théorème 17

Si d divise d' et d' divise d alors $d' = d$

Théorème 16

Si d divise a alors $|d| \leqslant |a|$.

Théorème 17

Si d divise d' et d' divise d alors $d' = d$.

Théorème 16

Si d divise a alors $|d| \leqslant |a|$.

Théorème 17

Si d divise d' et d' divise d alors $d' = d$.

Théorème 18 (Toute suite strictement décroissante d'entiers naturels ...)

Toute suite strictement décroissante d'entiers naturels est constante à partir d'un certain rang.

Théorème 16

Si d divise a alors $|d| \leqslant |a|$.

Théorème 17

Si d divise d' et d' divise d alors $d' = d$.

Théorème 18 (Suite strictement décroissante d'entiers naturels)

Toute suite strictement décroissante d'entiers naturels est constante à partir d'un certain rang.

Théorème 16

Si d divise a alors $|d| \leqslant |a|$.

Théorème 17

Si d divise d' et d' divise d alors $d' = d$.

Théorème 18 (Suite strictement décroissante d'entiers naturels)

Toute suite strictement décroissante d'entiers naturels est constante à partir d'un certain rang.

Sommaire

PGCD

Définition 19 (PGCD)

Soit a et b deux entiers. On appelle PGCD de a et b et on note $a \wedge b$ l'entier défini de la manière suivante :

- $0 \wedge 0 = 0$
- Si a et b ne sont pas simultanément nuls, $a \wedge b$ est le plus grand entier naturel qui divise simultanément a et b.

$$D(a) \cap D(b)$$

PGCD

Définition 19 (PGCD)

Soit a et b deux entiers. On appelle PGCD de a et b et on note $a \wedge b$ l'entier défini de la manière suivante :

- $0 \wedge 0 = 0$
- Si a et b ne sont pas simultanément nuls, $a \wedge b$ est le plus grand entier naturel qui divise simultanément a et b.

$$\mathcal{D}(a) \cap \mathcal{D}(b)$$

PGCD

Définition 19 (PGCD)

Soit a et b deux entiers. On appelle PGCD de a et b et on note $a \wedge b$ l'entier défini de la manière suivante :

- $0 \wedge 0 = 0$
- Si a et b ne sont pas simultanément nuls, $a \wedge b$ est le plus grand entier naturel qui divise simultanément a et b.

$$\mathcal{D}(a) \cap \mathcal{D}(b)$$

PGCD

Définition 19 (PGCD)

Soit a et b deux entiers. On appelle PGCD de a et b et on note $a \wedge b$ l'entier défini de la manière suivante :

- $0 \wedge 0 = 0$
- Si a et b ne sont pas simultanément nuls, $a \wedge b$ est le plus grand entier naturel qui divise simultanément a et b.

$$\mathcal{D}(a) \cap \mathcal{D}(b)$$

PGCD

Définition 19 (PGCD)

Soit a et b deux entiers. On appelle PGCD de a et b et on note $a \wedge b$ l'entier défini de la manière suivante :

- $0 \wedge 0 = 0$
- Si a et b ne sont pas simultanément nuls, $a \wedge b$ est le plus grand entier naturel qui divise simultanément a et b.

$$\mathcal{D}(a) \cap \mathcal{D}(b)$$

Théorème 20 (Égalité de Bézout)

Soit a et b deux entiers relatifs. Si $d = a \wedge b$, alors il existe deux entiers u et v tels que :

$$au + bv = d$$

Théorème 21

Si $ab \neq 0$ *et* k *un entier quelconque*

$$a \wedge (b + ka) = a \wedge b$$

et en particulier

$$a \wedge b = a \wedge (b - a) = a \wedge (a - b)$$

Algorithme des différences

Théorème 22

Si $ab \neq 0$ *et* k *un entier quelconque*

$$a \wedge (b + ka) = a \wedge b$$

et en particulier

$$a \wedge b = a \wedge (b - a) = a \wedge (a - b)$$

Algorithme d'Euclide I

Fonction `euclide(a, b:`*entiers*`):`*entier*

Si `b = 0` **Alors**

  Retourner `a`

Sinon

  Retourner `euclide(b,rem(a,b))`

FinSi

Algorithme d'Euclide II

k	u_k	v_k	r_k	q_k
0	1	0	$r_0 = a$	/
1	0	1	$r_1 = b$	q_1
2	$u_0 - u_1 q_1$	$v_0 - v_1 q_1$	$r_2 = r_0 - r_1 q_1$	q_2
3	$u_1 - u_2 q_2$	$v_1 - v_2 q_2$	$r_3 = r_1 - r_2 q_2$	q_3
$\vdots$			$\vdots$	$\vdots$
$p-1$			$r_{p-1} = a \wedge b$	q_{p-1}
p			$r_p = 0$	

k	u_k	v_k	r_k	q_k	
0	1	0	19	/	L_0
1	0	1	15	1	L_1
2	1	-1	4	3	$L_2 \leftarrow L_0 - 1 \times L_1$
3	-3	4	3	1	$L_3 \leftarrow L_1 - 3 \times L_2$
4	4	-5	1	3	$L_4 \leftarrow L_2 - 1 \times L_3$
5			0		$L_5 \leftarrow L_3 - 3 \times L_4$

Fonction `euc(u,v,r,u',v',r'` : *entiers*`)` : *liste d'entiers*

Si `r' = 0` **Alors**

  Retourner `[u,v,r]`

Sinon

  Retourner

```
euc(u',v',r',u-quo(r,r')*u',v-quo(r,r')*v',r-quo(r,r')*r')
```

FinSi

Fonction `EUC(a,b` : *entiers*`)` : *liste d'entiers*

Retourner `euc(1,0,a,0,1,b)`

Nombres premiers entre eux

Définition 23

Deux entiers a et b sont premiers entre eux si et seulement si

$$a \wedge b = 1$$

Théorème 24 (Théorème de Bézout)

Les entiers a et b sont premiers entre eux si, et seulement si, il existe deux entiers u et v tels que $au + bv = 1$

$$a \wedge b = 1 \iff \text{il existe } (u, v) \in \mathbb{Z}^2 \text{ tel que } au + bv = 1$$

Nombres premiers entre eux

Définition 23

Deux entiers a et b sont premiers entre eux si et seulement si

$$a \wedge b = 1$$

Théorème 24 (Théorème de Bézout)

Les entiers a et b sont premiers entre eux si, et seulement si, il existe deux entiers u et v tels que $au + bv = 1$

$$a \wedge b = 1 \iff \text{il existe } (u, v) \in \mathbb{Z}^2 \text{ tel que } au + bv = 1$$

Nombres premiers entre eux

Définition 23

Deux entiers a et b sont premiers entre eux si et seulement si

$$a \wedge b = 1$$

Théorème 24 (Théorème de Bézout)

Les entiers a et b sont premiers entre eux si, et seulement si, il existe deux entiers u et v tels que $au + bv = 1$

$$a \wedge b = 1 \iff \text{il existe } (u, v) \in \mathbb{Z}^2 \text{ tel que } au + bv = 1$$

Sommaire

Définition 25

Un entier naturel est dit premier s'il est supérieur (i.e. supérieur ou égal) à 2 et n'est divisible que par 1 et lui-même.

Définition 25

Un entier naturel est dit premier s'il est supérieur (i.e. supérieur ou égal) à 2 et n'est divisible que par 1 et lui-même.

Définition 25

Un entier naturel est dit premier s'il est supérieur (i.e. supérieur ou égal) à 2 et n'est divisible que par 1 et lui-même.

diviseur	2	3	4	5	6	7	8	9	10	11	12	13
quotient	660,5	440,3	330,3	264,2	220,2	188,7	165,1	146,8	132,1	120,1	110,1	101,6
diviseur	14	15	16	17	18	19	20	21	22	23	24	25
quotient	94,36	88,07	82,56	77,71	73,39	69,53	66,05	62,90	60,05	57,43	55,04	52,84
diviseur	26	27	28	29	30	31	32	33	34	35	36	37
quotient	50,81	48,93	47,18	45,55	44,03	42,61	41,28	40,03	38,85	37,74	36,69	35,70

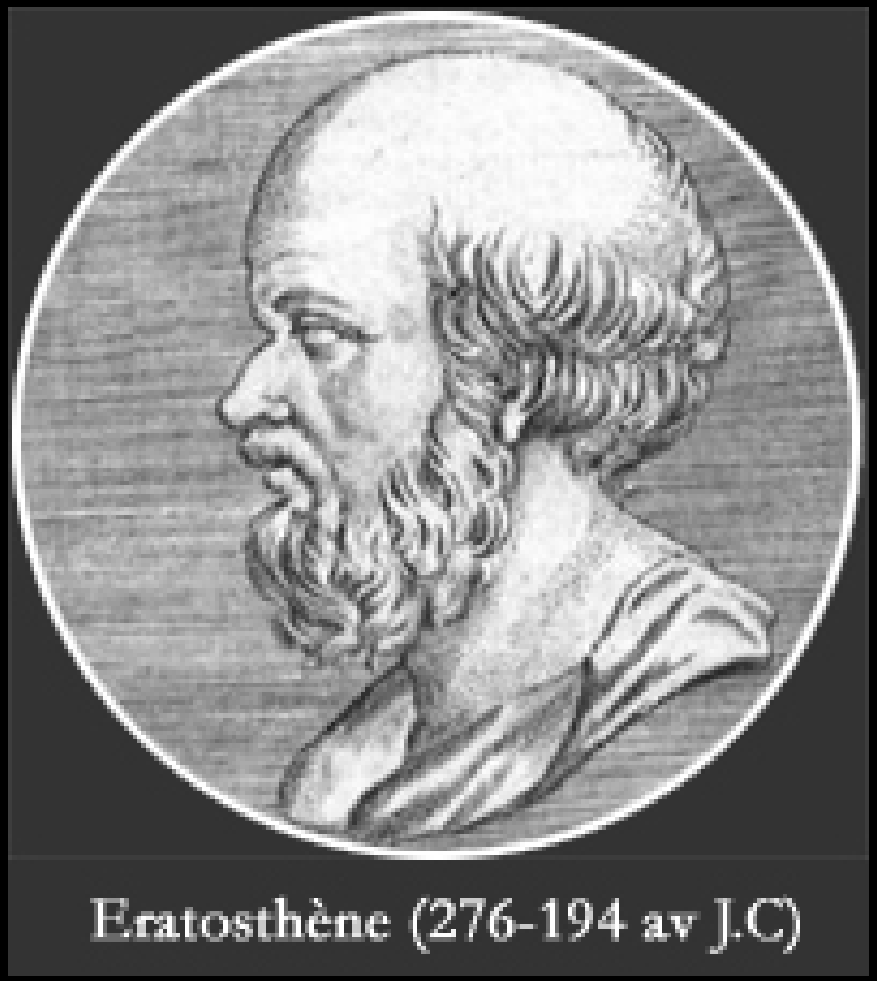
Théorème 26

Si un entier est composé, alors il admet un diviseur premier inférieur à sa racine carrée.

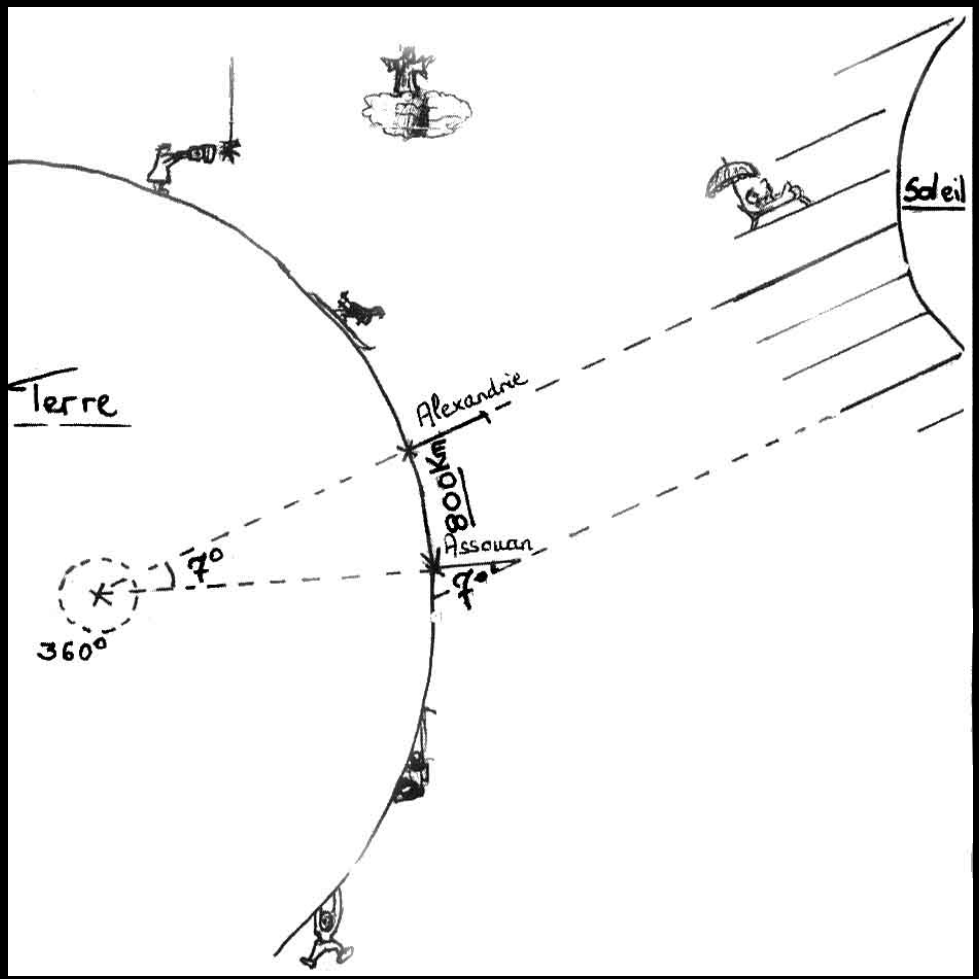
diviseur	2	3	4	5	6	7	8	9	10	11	12	13
quotient	660,5	440,3	330,3	264,2	220,2	188,7	165,1	146,8	132,1	120,1	110,1	101,6
diviseur	14	15	16	17	18	19	20	21	22	23	24	25
quotient	94,36	88,07	82,56	77,71	73,39	69,53	66,05	62,90	60,05	57,43	55,04	52,84
diviseur	26	27	28	29	30	31	32	33	34	35	36	37
quotient	50,81	48,93	47,18	45,55	44,03	42,61	41,28	40,03	38,85	37,74	36,69	35,70

Théorème 26

Si un entier est composé, alors il admet un diviseur premier inférieur à sa racine carrée.



Eratosthène (276-194 av J.C)


Soleil
Terre
Alexandrie
800 km
Assouan
7°
7°
360°

```
val filter : ('a -> bool) -> 'a list -> 'a list

filter p l returns all the elements of the list l that satisfy the
    predicate p. The order of the elements in the input list is preserved
    .
```

```
let retmultiples = fun liste n -> ...
```

```
# retmultiples [2;3;4;5;6;7;8;9;10;11;12] 3 ;;
- : int list = [2; 4; 5; 7; 8; 10; 11]
```

```
val filter : ('a -> bool) -> 'a list -> 'a list

filter p l returns all the elements of the list l that satisfy the
    predicate p. The order of the elements in the input list is preserved
    .
```

```
let retmultiples = fun liste n -> ...
```

```
# retmultiples [2;3;4;5;6;7;8;9;10;11;12] 3 ;;
- : int list = [2; 4; 5; 7; 8; 10; 11]
```

```
val filter : ('a -> bool) -> 'a list -> 'a list

filter p l returns all the elements of the list l that satisfy the
    predicate p. The order of the elements in the input list is preserved
    .
```

```
let retmultiples = fun liste n -> ...
```

```
# retmultiples [2;3;4;5;6;7;8;9;10;11;12] 3 ;;
- : int list = [2; 4; 5; 7; 8; 10; 11]
```

```
let liste_entiers = fun min max ->
  let rec aux = fun a acc ->
    if ...
    else ...
  in aux min [];;
```

```
let crible = fun m ->
  let rec crible_rec = fun n acc ->
      if n*n > m then
        acc
      else
        crible_rec (n+1) (retmultiples acc n)
  in crible_rec 2 (liste_entiers 2 m);;
```

```
# crible 100;;
- : int list =
[97; 89; 83; 79; 73; 71; 67; 61; 59; 53; 47; 43; 41; 37; 31; 29; 23; 19;
    17; 13; 11; 7; 5; 3; 2]
```

```
let crible = fun m ->
  let rec crible_rec = fun n acc ->
      if n*n > m then
        acc
      else
        crible_rec (n+1) (retmultiples acc n)
  in crible_rec 2 (liste_entiers 2 m);;
```

```
# crible 100;;
- : int list =
[97; 89; 83; 79; 73; 71; 67; 61; 59; 53; 47; 43; 41; 37; 31; 29; 23; 19;
    17; 13; 11; 7; 5; 3; 2]
```

Théorème 27

Tout entier n supérieur à 2 admet une et une seule (à l'ordre près des termes) décomposition en produit fini de nombres premiers

Sommaire

Définition 28 (Anneau)

Soit A un ensemble muni de deux lois $\cdot$ et $+$ qu'on nommera addition et multiplication.
On dit que $(A, +, \cdot)$ est un anneau si, et seulement si :

- $(A, +)$ est un groupe commutatif ;
- la multiplication est associative ;
- la multiplication est distributive sur l'addition.

Si la multiplication est commutative, alors l'anneau est dit *commutatif*.
Si la multiplication admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par la multiplication sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 28 (Anneau)

Soit A un ensemble muni de deux lois $\cdot$ et $+$ qu'on nommera addition et multiplication.
On dit que $(A, +, \cdot)$ est un anneau si, et seulement si :

- $(A, +)$ est un groupe commutatif ;
- la multiplication est associative ;
- la multiplication est distributive sur l'addition.

Si la multiplication est commutative, alors l'anneau est dit *commutatif*.
Si la multiplication admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par la multiplication sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 28 (Anneau)

Soit A un ensemble muni de deux lois $\cdot$ et $+$ qu'on nommera addition et multiplication.
On dit que $(A, +, \cdot)$ est un anneau si, et seulement si :

- $(A, +)$ est un groupe commutatif ;
- la multiplication est associative ;
- la multiplication est distributive sur l'addition.

Si la multiplication est commutative, alors l'anneau est dit *commutatif*.
Si la multiplication admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par la multiplication sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 28 (Anneau)

Soit A un ensemble muni de deux lois $\cdot$ et $+$ qu'on nommera addition et multiplication.
On dit que $(A, +, \cdot)$ est un anneau si, et seulement si :

- $(A, +)$ est un groupe commutatif ;
- la multiplication est associative ;
- la multiplication est distributive sur l'addition.

Si la multiplication est commutative, alors l'anneau est dit *commutatif*.
Si la multiplication admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par la multiplication sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Théorème 29

$\left((\mathbb{Z}/n\mathbb{Z})^*, \cdot\right)$ *est un groupe.*

Théorème 30

- *Les seuls éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$ sont les éléments premiers avec n.*
- *Si n est premier, $\mathbb{Z}/n\mathbb{Z}$ est un corps et on le note dans ce cas $\mathbb{F}_n$.*

Théorème 30

- *Les seuls éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$ sont les éléments premiers avec n.*
- *Si n est premier, $\mathbb{Z}/n\mathbb{Z}$ est un corps et on le note dans ce cas $\mathbb{F}_n$.*

Théorème 30

- *Les seuls éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$ sont les éléments premiers avec n.*
- *Si n est premier, $\mathbb{Z}/n\mathbb{Z}$ est un corps et on le note dans ce cas $\mathbb{F}_n$.*

Définition 31 (Fonction indicatrice d'Euler)

On note $\varphi(n)$ le nombre d'éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$. C'est donc aussi le cardinal (on dit aussi l'*ordre*) du groupe $\left((\mathbb{Z}/n\mathbb{Z})^*, \cdot\right)$.

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

- *Si p est premier, alors $\varphi(p) = p - 1$;*
- *si $m \wedge n = 1$ alors $\varphi(mn) = \varphi(m)\varphi(n)$ (admis) ;*
- *Si p est premier, $\varphi(p^n) = p^n - p^{n-1} = p^n\left(1 - \frac{1}{p}\right)$;*
- *$\varphi(n) = n \prod_{p \in P_n} \left(1 - \frac{1}{p}\right)$ avec P_n l'ensemble des diviseurs premiers de n.*

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

- *Si p est premier, alors $\varphi(p) = p - 1$;*
- *si $m \wedge n = 1$ alors $\varphi(mn) = \varphi(m)\varphi(n)$ (admis) ;*
- *Si p est premier, $\varphi(p^n) = p^n - p^{n-1} = p^n\left(1 - \frac{1}{p}\right)$;*
- $\varphi(n) = n \prod_{p \in \mathcal{P}_n} \left(1 - \frac{1}{p}\right)$ *avec $\mathcal{P}_n$ l'ensemble des diviseurs premiers de n.*

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

- *Si p est premier, alors $\varphi(p) = p - 1$;*
- *si $m \wedge n = 1$ alors $\varphi(mn) = \varphi(m)\varphi(n)$ (admis) ;*
- *Si p est premier, $\varphi(p^n) = p^n - p^{n-1} = p^n\left(1 - \frac{1}{p}\right)$;*
- *$\varphi(n) = n \prod_{p \in \mathcal{P}_n} \left(1 - \frac{1}{p}\right)$ avec $\mathcal{P}_n$ l'ensemble des diviseurs premiers de n.*

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

- *Si p est premier, alors $\varphi(p) = p - 1$;*
- *si $m \wedge n = 1$ alors $\varphi(mn) = \varphi(m)\varphi(n)$ (admis) ;*
- *Si p est premier, $\varphi(p^n) = p^n - p^{n-1} = p^n \left(1 - \frac{1}{p}\right)$;*
- *$\varphi(n) = n \prod_{p \in \mathcal{P}_n} \left(1 - \frac{1}{p}\right)$ avec $\mathcal{P}_n$ l'ensemble des diviseurs premiers de n.*

Théorème 32 (Théorème d'Euler)

$$a \wedge n = 1 \Longrightarrow a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Théorème 33 (Propriétés de $\varphi(n)$)

- *Si p est premier, alors $\varphi(p) = p - 1$;*
- *si $m \wedge n = 1$ alors $\varphi(mn) = \varphi(m)\varphi(n)$ (admis) ;*
- *Si p est premier, $\varphi(p^n) = p^n - p^{n-1} = p^n \left(1 - \frac{1}{p}\right)$;*
- $\varphi(n) = n \prod_{p \in \mathcal{P}_n} \left(1 - \frac{1}{p}\right)$ *avec $\mathcal{P}_n$ l'ensemble des diviseurs premiers de n.*

Théorème 34 (Petit théorème de Fermat)

- *Si p est premier et ne divise pas a alors $a^{p-1} \equiv 1 (\mathrm{mod}\, p)$;*
- *Si p est premier, $a^p \equiv a (\mathrm{mod}\, p)$.*

Théorème 34 (Petit théorème de Fermat)

- *Si p est premier et ne divise pas a alors* $a^{p-1} \equiv 1 (\mathrm{mod}\, p)$;
- *Si p est premier,* $a^p \equiv a (\mathrm{mod}\, p)$.

Théorème 34 (Petit théorème de Fermat)

- *Si p est premier et ne divise pas a alors* $a^{p-1} \equiv 1 (\mathrm{mod}\, p)$;
- *Si p est premier,* $a^p \equiv a (\mathrm{mod}\, p)$.

$x^n + y^n = z^n$ n'a pas de solution dans $\mathbb{N}^3$ pour $n > 2$

$x^n + y^n = z^n$ n'a pas de solution dans $\mathbb{N}^3$ pour $n > 2$

Sommaire

Sommaire


VA A ROME PORTER UN MESSAGE À CÉSAR. TU LUI DIRAS : " TOUTE LA GAULE EST OCCUPÉE ", IL TE DEMANDERA : "TOUTE ?", TU LUI RÉPONDRAS : "TOUTE !" IL COMPRENDRA.
Uderzo & Goscinny

Comme le disait Suétone (70-127) dans La vie des 12 Césars :

> *Extant et ad Ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum uerbum effici posset : quae si qui inuestigare et persequi uelit, quartam elementorum litteram, id est D pro A et perinde reliquas commutet.*

Ce que César aurait peut-être écrit sous cette forme :

> *Rkgnag rg nq Pvprebarz, vgrz nq snzvyvnerf qbzrfgvpvf qr erohf, va dhvohf, fv dhn bpphygvhf cresreraqn renag, cre abgnf fpevcfvg, vq rfg fvp fgehpgb yvggrenehz beqvar, hg ahyyhz hreohz rssvpv cbffrg : dhnr fv dhv vahrfgvtner rg crefrdhv hryvg, dhnegnz ryrzragbehz yvggrenz, vq rfg Q ceb N rg crevaqr eryvdhnf pbzzhgrg.*

Ou sous celle-ci si Rome avait été colonisé par des informaticiens américains :

"H{wdqw#hw#dg#Flfhurqhp/#lwhp#dg#idplolduhv#grphvwlflv#gh#uhexv/#lq#txlexv/#vl#txd#rffxowlxv#shuihuhqgd#hudqw/#shu#qrwdv#vfulsvlw/#lg#hvw#vlf#vwuxfwr#olwwhuduxp#ruglqh/#xw#qxooxp#xhuexp#hiilfl#srvvhw=#txdh#vl#txl#lqxhvwljduh#hw#shuvhtxl#xholw/#txduwdp#hohphqwruxp#olwwhudp/#lg#hvw#G#sur#D#hw#shulqgh#uholtxdv#frppxwhw1"

Ceux qui ne font pas de latin de spécialité seront sûrement plus à l'aise avec cette nouvelle transcription du même texte :

> *On possède enfin de César des lettres à Cicéron, et sa correspondance avec ses amis sur ses affaires domestiques. Il y employait, pour les choses tout à fait secrètes, une espèce de chiffre qui en rendait le sens inintelligible (les lettres étant disposées de manière à ne pouvoir jamais former un mot), et qui consistait, je le dis pour ceux qui voudront les déchiffrer, à changer le rang des lettres dans l'alphabet, en écrivant la quatrième pour la première, c'est-à-dire le D pour l'A, et ainsi de suite.*

Vous êtes César

Vous voulez transmettre cet important message :

Les sanglots longs des violons de l'automne

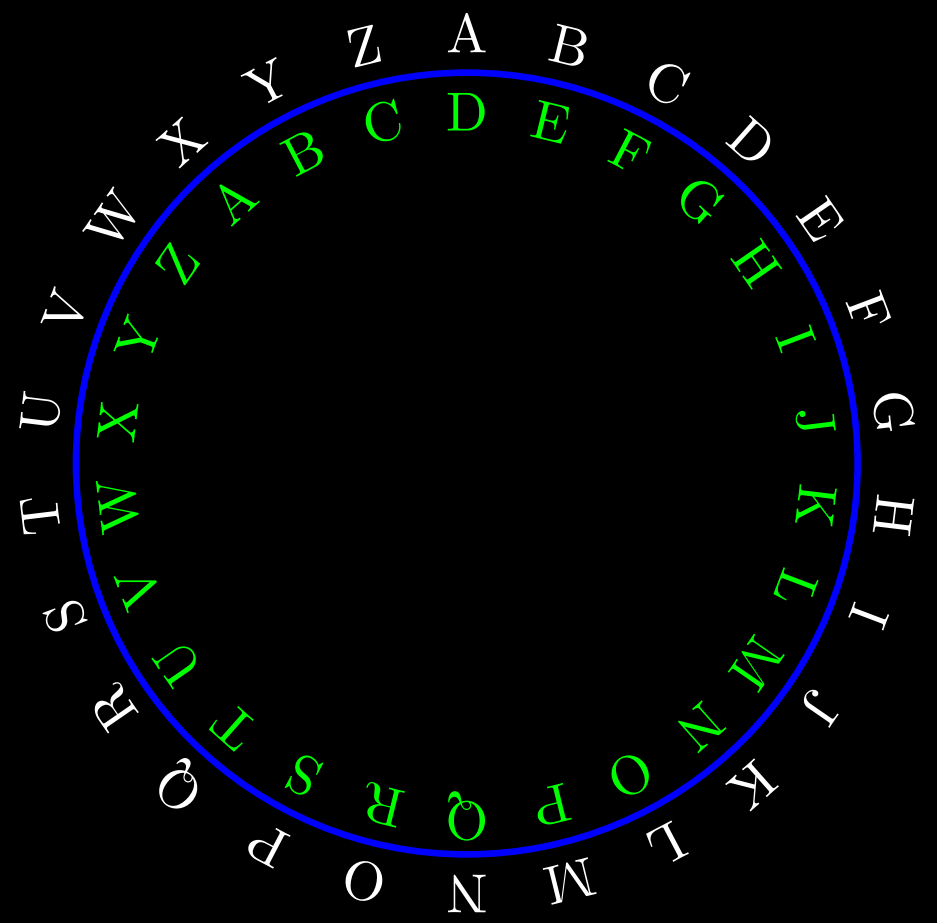
Vous êtes Vercingétorix

Vous voulez traduire ce message intercepté par vos espions :

*eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh*

La règle ou la roue ?

Que vous inspire ce dessin :



```
# List.map int_of_char ['a';'b';'c';'d';'e';'f';'g';'h'];;
- : int list = [97; 98; 99; 100; 101; 102; 103; 104]
```

```
# List.map char_of_int [65; 66; 67; 68; 69; 70; 71; 72];;
- : char list = ['A'; 'B'; 'C'; 'D'; 'E'; 'F'; 'G'; 'H']
```

```
# List.map int_of_char ['a';'b';'c';'d';'e';'f';'g';'h'];;
- : int list = [97; 98; 99; 100; 101; 102; 103; 104]
```

```
# List.map char_of_int [65; 66; 67; 68; 69; 70; 71; 72];;
- : char list = ['A'; 'B'; 'C'; 'D'; 'E'; 'F'; 'G'; 'H']
```

Caractère		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2
Code	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Caractère	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E
Code	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
Caractère	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Code	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88
Caractère	Y	Z	[	\	]	^	_	‘	a	b	c	d	e	f	g	h	i	j	k
Code	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
Caractère	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~
Code	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126

Table des caractères ASCII affichables

```
# decalage 3 97;;
- : int = 100
```

```
# decalage 3 125;;
- : int = 33
```

```
# code 3 'a';;
- : char = 'd'
```

```
# code 3 '}';;
- : char = '!'
```

```
# decalage 3 97;;
- : int = 100
```

```
# decalage 3 125;;
- : int = 33
```

```
# code 3 'a';;
- : char = 'd'
```

```
# code 3 '}';;
- : char = '!'
```

```
# decalage 3 97;;
- : int = 100
```

```
# decalage 3 125;;
- : int = 33
```

```
# code 3 'a';;
- : char = 'd'
```

```
# code 3 '}';;
- : char = '!'
```

```
# decalage 3 97;;
- : int = 100
```

```
# decalage 3 125;;
- : int = 33
```

```
# code 3 'a';;
- : char = 'd'
```

```
# code 3 '}';;
- : char = '!'
```

```
# cesar 3 "Les sanglots longs des violons de l'automne";;
- : string = "Ohv#vdqjorwv#orqjv#ghv#ylrorqv#gh#o*dxwrpqh"
```

et pour déchiffrer

« *eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh* » ?

```
# cesar (-3) "eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh";;
- : string = "blessent mon coeur d'une langueur monotone"
```

```
# String.map;;
- : (char -> char) -> string -> string = <fun>
```

```
# cesar 3 "Les sanglots longs des violons de l'automne";;
- : string = "Ohv#vdqjorwv#orqjv#ghv#ylrorqv#gh#o*dxwrpqh"
```

et pour déchiffrer

« *eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh* » ?

```
# cesar (-3) "eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh";;
- : string = "blessent mon coeur d'une langueur monotone"
```

```
# String.map;;
- : (char -> char) -> string -> string = <fun>
```

```
# cesar 3 "Les sanglots longs des violons de l'automne";;
- : string = "Ohv#vdqjorwv#orqjv#ghv#ylrorqv#gh#o*dxwrpqh"
```

et pour déchiffrer
« *eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh* » ?

```
# cesar (-3) "eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh";;
- : string = "blessent mon coeur d'une langueur monotone"
```

```
# String.map;;
- : (char -> char) -> string -> string = <fun>
```

```
# cesar 3 "Les sanglots longs des violons de l'automne";;
- : string = "Ohv#vdqjorwv#orqjv#ghv#ylrorqv#gh#o*dxwrpqh"
```

et pour déchiffrer

« *eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh* » ?

```
# cesar (-3) "eohvvhqw#prq#frhxu#g*xqh#odqjxhxu#prqrwrqh";;
- : string = "blessent mon coeur d'une langueur monotone"
```

```
# String.map;;
- : (char -> char) -> string -> string = <fun>
```

Sommaire

Ce n'est pas trop dur de casser le code de César. Nous allons essayer de faire mieux en utilisant nos outils arithmétiques.

On considère toujours les 95 caractères affichables et on les associe aux nombres 0, 1, 2,...,94.

On code ces nombres à l'aide d'une fonction f du type :

$$f : x \mapsto (ax + b) \bmod 95$$

où a et b sont des entiers.

Quid du code de César ?

Ce n'est pas trop dur de casser le code de César. Nous allons essayer de faire mieux en utilisant nos outils arithmétiques.
On considère toujours les 95 caractères affichables et on les associe aux nombres 0, 1, 2,...,94.
On code ces nombres à l'aide d'une fonction f du type :

$$f : x \mapsto (ax + b) \bmod 95$$

où a et b sont des entiers.
Quid du code de César ?

Ce n'est pas trop dur de casser le code de César. Nous allons essayer de faire mieux en utilisant nos outils arithmétiques.
On considère toujours les 95 caractères affichables et on les associe aux nombres 0, 1, 2,...,94.
On code ces nombres à l'aide d'une fonction f du type :

$$f : \ x \mapsto (ax + b) \bmod 95$$

où a et b sont des entiers.
Quid du code de César ?

Ce n'est pas trop dur de casser le code de César. Nous allons essayer de faire mieux en utilisant nos outils arithmétiques.
On considère toujours les 95 caractères affichables et on les associe aux nombres 0, 1, 2,...,94.
On code ces nombres à l'aide d'une fonction f du type :

$$f : \ x \mapsto (ax + b) \bmod 95$$

où a et b sont des entiers.
Quid du code de César ?

```
# chiffre_affine 17 22 "abcd";;
- : string = "r$5F"
```

Comment décrypter un message ? On cherche g telle que :

$$\text{transmis} = f(\text{original}) \Leftrightarrow \text{original} = g(\text{transmis}) \text{ dans } \mathbb{Z}_{95}$$

On a donc besoin de connaître l'inverse de a dans $\mathbb{Z}_{95}$...en s'assurant qu'il existe !

```
# chiffre_affine 17 22 "abcd";;
- : string = "r$5F"
```

Comment décrypter un message ? On cherche g telle que :

$$\text{transmis} = f(\text{original}) \Leftrightarrow \text{original} = g(\text{transmis}) \text{ dans } \mathbb{Z}_{95}$$

On a donc besoin de connaître l'inverse de a dans $\mathbb{Z}_{95}$...en s'assurant qu'il existe !

```
# chiffre_affine 17 22 "abcd";;
- : string = "r$5F"
```

Comment décrypter un message ? On cherche g telle que :

$$\text{transmis} = f(\text{original}) \Leftrightarrow \text{original} = g(\text{transmis}) \text{ dans } \mathbb{Z}_{95}$$

On a donc besoin de connaître l'inverse de a dans $\mathbb{Z}_{95}$...en s'assurant qu'il existe !

```
# chiffre_affine 17 22 "abcd";;
- : string = "r$5F"
```

Comment décrypter un message ? On cherche g telle que :

$$\text{transmis} = f(\text{original}) \Leftrightarrow \text{original} = g(\text{transmis}) \text{ dans } \mathbb{Z}_{95}$$

On a donc besoin de connaître l'inverse de a dans $\mathbb{Z}_{95}$...en s'assurant qu'il existe !

```
# chiffre_affine 17 22 "aaaa";;
- : string = "rrrr"
```

Sommaire

Sommaire

Définition 35 (Permutation)

Soit E un ensemble. Une permutation de E est une application bijective de E sur E. On note $\mathfrak{S}(E)$ l'ensemble des permutations de E.

Définition 36

Soit $n \in \mathbb{N}$, alors $\mathfrak{S}_n$ désigne l'ensemble des permutations de $\{1, 2, 3, \ldots, n\}$

Définition 35 (Permutation)

Soit E un ensemble. Une permutation de E est une application bijective de E sur E. On note $\mathfrak{S}(E)$ l'ensemble des permutations de E.

Définition 36

Soit $n \in \mathbb{N}$, alors $\mathfrak{S}_n$ désigne l'ensemble des permutations de $\{1, 2, 3, ..., n\}$.

Définition 35 (Permutation)

Soit E un ensemble. Une permutation de E est une application bijective de E sur E. On note $\mathfrak{S}(E)$ l'ensemble des permutations de E.

Définition 36

Soit $n \in \mathbb{N}$, alors $\mathfrak{S}_n$ désigne l'ensemble des permutations de $\{1, 2, 3, ..., n\}$.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 2 & 5 & 4 \end{pmatrix}$$

Théorème 37 (Groupe des permutations)

L'ensemble $\mathfrak{S}(E)$ des permutations sur un ensemble E, muni de la loi $\circ$ de composition des applications, a une structure de groupe.

Théorème 38 (Nombre de permutations)

Le groupe $\mathfrak{S}_n$ est d'ordre $n!$.

Théorème 37 (Groupe des permutations)

L'ensemble $\mathfrak{S}(E)$ des permutations sur un ensemble E, muni de la loi $\circ$ de composition des applications, a une structure de groupe.

Théorème 38 (Nombre de permutations)

Le groupe $\mathfrak{S}_n$ est d'ordre $n!$.

Une *permutation circulaire gauche* « décale » les éléments d'un nombre fixé de « rangs ».

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix}$$

Une *permutation circulaire gauche* « décale » les éléments d'un nombre fixé de « rangs ».

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix}$$

Sommaire

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.

Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.

Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.

Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.

Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.
Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.
Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.
Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.

Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Définition 39 (Cryptosystème (ou système de chiffrement ou chiffre))

Un cryptosystème est un quintuplet $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ tel que :

- L'ensemble $\mathcal{P}$ est l'espace des messages en clair (*plaintext* en anglais) ;
- L'ensemble $\mathcal{C}$ est l'espace des messages chiffrés (*cyphertext* en anglais) ;
- L'ensemble $\mathcal{K}$ est l'espace des clés (*key*) ;
- $\mathcal{E}$ est la famille des fonctions de chiffrement, qui vont de $\mathcal{P}$ dans $\mathcal{C}$;
- $\mathcal{D}$ est la famille des fonctions de déchiffrement, qui vont de $\mathcal{C}$ dans $\mathcal{P}$;

Pour chaque élément $e \in \mathcal{K}$, il existe un élément $d \in \mathcal{K}$ tel que, pour tout message clair m de $\mathcal{P}$, il existe $D_d \in \mathcal{D}$ et $E_e \in \mathcal{E}$ telles que $D_d(E_e(m)) = m$. Les fonctions D_d et E_e sont des applications injectives. Il est entendu que d doit rester secret...

Un cryptosystème est un *chiffrement par blocs* si $\mathcal{P} = \mathcal{C} = A_n$ avec A_n l'ensemble des mots de l'alphabet A de longueur n.
Depuis 2001, le chiffrement par blocs « officiel » est l'AES qui opère par blocs de 128 bits.

Sommaire

Electronic CodeBook

On considère une chaîne de bits quelconque que l'on découpe en blocs de longueur fixe, par exemple 7 (plus pratique pour ensuite utiliser l'ASCII : pourquoi ?). On rajoute éventuellement des zéros en bout de chaîne.

Electronic CodeBook

On considère une chaîne de bits quelconque que l'on découpe en blocs de longueur fixe, par exemple 7 (plus pratique pour ensuite utiliser l'ASCII : pourquoi ?). On rajoute éventuellement des zéros en bout de chaîne.

Considérons $m = 1001001111011$.

On rajoute un 0 en bout de chaîne :

$$m' = 1001001\ 1110110 = \beta_1\beta_2$$

avec $\beta_1 = 1001001$ et $\beta_2 = 1110110$.

On choisit une clé de chiffrement dans $\mathfrak{S}_7$ car $\mathcal{K} = \mathfrak{S}_7$:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 4 & 2 & 6 & 5 & 7 \end{pmatrix}$$

Alors $E_\pi(\beta_1) = 0110001$ et $E_\pi(\beta_2) = 1101110$.

Considérons $m = 1001001111011$.
On rajoute un 0 en bout de chaîne :

$$m' = 1001001\ 1110110 = \beta_1\beta_2$$

avec $\beta_1 = 1001001$ et $\beta_2 = 1110110$.
On choisit une clé de chiffrement dans $\mathfrak{S}_7$ car $\mathcal{K} = \mathfrak{S}_7$:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 4 & 2 & 6 & 5 & 7 \end{pmatrix}$$

Alors $E_\pi(\beta_1) = 0110001$ et $E_\pi(\beta_2) = 1101110$.

Considérons $m = 1001001111011$.
On rajoute un 0 en bout de chaîne :

$$m' = 1001001\ 1110110 = \beta_1\beta_2$$

avec $\beta_1 = 1001001$ et $\beta_2 = 1110110$.
On choisit une clé de chiffrement dans $\mathfrak{S}_7$ car $\mathcal{K} = \mathfrak{S}_7$:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 4 & 2 & 6 & 5 & 7 \end{pmatrix}$$

Alors $E_\pi(\beta_1) = 0110001$ et $E_\pi(\beta_2) = 1101110$.

Considérons $m = 1001001111011$.
On rajoute un 0 en bout de chaîne :

$$m' = 1001001\ 1110110 = \beta_1\beta_2$$

avec $\beta_1 = 1001001$ et $\beta_2 = 1110110$.
On choisit une clé de chiffrement dans $\mathfrak{S}_7$ car $\mathcal{K} = \mathfrak{S}_7$:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 4 & 2 & 6 & 5 & 7 \end{pmatrix}$$

Alors $E_\pi(\beta_1) = 0110001$ et $E_\pi(\beta_2) = 1101110$.

Sommaire

CBC : *Cipher-Block Chaining*. Ce mode a été inventé par Ibm en 1976. Pour éviter la faiblesse de l'ECB, les blocs sont maintenant chaînés : chaque bloc en clair est « XORé » ou « OUEXé » avec le bloc crypté précédent avant d'être crypté lui-même.

Pour le premier bloc, on utilise un *vecteur d'initialisation* public. Avec les notations habituelles, on a donc

$$c_i = E_k(m_i \oplus c_{i-1}), \quad c_0 = \overrightarrow{v_0}$$

CBC : *Cipher-Block Chaining*. Ce mode a été inventé par Ibm en 1976.
Pour éviter la faiblesse de l'ECB, les blocs sont maintenant chaînés : chaque bloc en clair est « XORé » ou « OUEXé » avec le bloc crypté précédent avant d'être crypté lui-même.
Pour le premier bloc, on utilise un *vecteur d'initialisation* public. Avec les notations habituelles, on a donc

$$c_i = E_k(m_i \oplus c_{i-1}), \quad c_0 = \overrightarrow{v_0}$$

Sommaire

CFB : *Cipher-FeedBack*. C'est un mode dérivé de CFB qui est utilisé par OpenPGP, format pour l'échange sécurisé de données (paiements sécurisés par exemple) largement utilisé actuellement mais est susceptible d'être attaqué, même s'il est très difficile de mettre en œuvre concrètement cette attaque.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.

On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- c_j est la r-ième reconstruction des r bits les plus à gauche de $b_j \oplus s_j$;
- t_{j+1} est le résultat du décalage de t_j de r bits vers la gauche, suivi de l'insertion de c_j dans les r bits de droite.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Le mode CFB utilise un registre de décalage de taille r inférieure à celle de la clé.
On a besoin d'un vecteur d'initialisation $\overrightarrow{v_0}$. Le message clair est découpé en blocs de longueur r. Ensuite on procède comme suit, sachant que les b_j sont les blocs en clair de longueur n (celle de la clé) et m_j sont les blocs en clair de longueur r :

- $t_0 = \overrightarrow{v_0}$;
- $s_j = E_k(t_j)$;
- g_j est la chaîne constituée des r bits les plus à gauche de s_j ;
- $c_j = m_j \oplus g_j$;
- $t_j = (2^r t_{j-1} + c_{j-1}) \bmod 2^n$.

Le déchiffrement fonctionne de manière identique en échangeant les rôles de m_j et c_j à la quatrième étape.

Sommaire

```
exception Non_defini;;

let pi = function
  |1 -> 3
  |2 -> 1
  |3 -> 4
  |4 -> 2
  |5 -> 6
  |6 -> 5
  |7 -> 7
  |_ -> raise Non_defini;;
```

```
(* l'inverse de pi *)
let pi_inv =
   true
;;
```

```
exception Non_defini;;

let pi = function
  |1 -> 3
  |2 -> 1
  |3 -> 4
  |4 -> 2
  |5 -> 6
  |6 -> 5
  |7 -> 7
  |_ -> raise Non_defini;;
```

```
(* l'inverse de pi *)
let pi_inv =
   true
;;
```

```
(* quid ? *)
let  liste_n = fun n ->
  let rec listentier = fun (m,n,acc) ->
    if m > n then
      List.rev acc
    else
      listentier (m+1,n,m::acc)
  in listentier (1,n,[]);;
```

```
(* quid ? *)
let permute = fun e liste ->
  List.rev (List.fold_left2 (fun acc el rg -> (List.nth liste ((e rg) - 1)
      )::acc ) [] liste (liste_n (List.length liste)));;
```

```
(* quid ?*)
let ecb = fun e liste ->
  List.map (permute e) liste;;
```

```
(* quid ? *)
let permute = fun e liste ->
  List.rev (List.fold_left2 (fun acc el rg -> (List.nth liste ((e rg) - 1)
      )::acc ) [] liste (liste_n (List.length liste)));;
```

```
(* quid ?*)
let ecb = fun e liste ->
  List.map (permute e) liste;;
```

```
(* quid ? *)
let paquets = fun n liste ->
  let rec aux = fun i li acc acc_tmp ->
    match li with
    |[]   -> if (i mod n = 0) then List.rev ((List.rev acc_tmp)::acc)
             else aux (i+1) ([]) acc (0::acc_tmp)
    |t::q -> if i = 0 then aux 1 q [] [t]
             else if i mod n = 0 then
               aux (i+1) q ((List.rev acc_tmp)::acc) [t]
                  else aux (i+1) q acc (t::acc_tmp)
  in aux 0 liste [] [];;
```

```
(* quid ? *)
let explose = fun chaine ->
  let rec aux = fun ch acc ->
    match ch with
     |"" -> List.rev acc
     |_  -> aux (String.sub ch 1 (String.length ch -1)) ((ch.[0])::acc)
  in aux chaine [];;
```

```
(* quid ? *)
let chaine_to_int_list = fun chaine ->
  let ex = explose chaine in
  List.flatten (List.map (base2_of_10) (List.map int_of_char ex));;
```

```
(* quid ? *)
let explose = fun chaine ->
  let rec aux = fun ch acc ->
    match ch with
     |"" -> List.rev acc
     |_  -> aux (String.sub ch 1 (String.length ch -1)) ((ch.[0])::acc)
  in aux chaine [];;
```

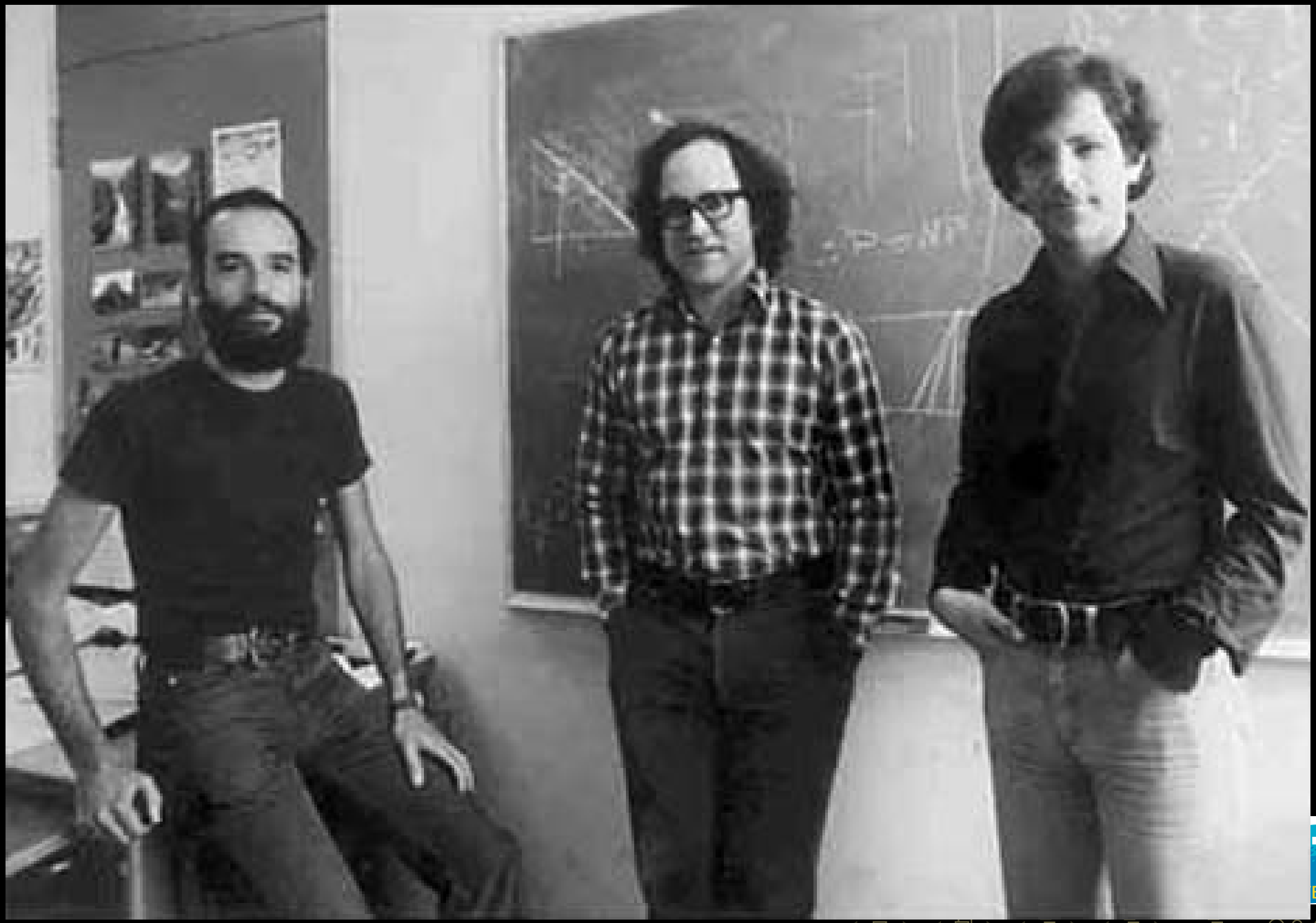
```
(* quid ? *)
let chaine_to_int_list = fun chaine ->
  let ex = explose chaine in
  List.flatten (List.map (base2_of_10) (List.map int_of_char ex));;
```

```
(* quid ? *)
let ecb_chaine = fun e chaine ->
  let liste_ini = paquets 7 (chaine_to_int_list chaine) in
  let liste_perm = paquets 7 (List.flatten (ecb e liste_ini)) in
  let liste_10 = List.map base10_of_2 liste_perm
  in List.map char_of_int liste_10;;
```

```
(* quid ? *)
let rec str_of_list = fun liste ->
  match liste with
   |[] -> ""
   |t::q -> (String.make 1 t)^(str_of_list q);;
```

Sommaire


rSa
Revenu de
solidarité
active


P=NP

- Votre amie Josette veut recevoir de vous une lettre d'amour mais elle a peur que le facteur l'intercepte et la lise.
- Elle fabrique donc dans son petit atelier une clé, un cadenas et une boîte vide .
- Elle vous envoie le cadenas, ouvert mais sans la clé, par la poste, donc à la merci du facteur : le cadenas est appelé *clé publique*.
- Vous recevez le cadenas et l'utilisez pour fermer une boîte contenant votre lettre : le facteur ne peut pas l'ouvrir car il n'a pas la clé.
- Josette reçoit donc la boîte fermée : elle utilise sa clé, qu'elle seule possède, pour ouvrir la boîte et se pâmer devant vos élans épistolaires.

- Votre amie Josette veut recevoir de vous une lettre d'amour mais elle a peur que le facteur l'intercepte et la lise.
- Elle fabrique donc dans son petit atelier une clé, un cadenas et une boîte vide .
- Elle vous envoie le cadenas, ouvert mais sans la clé, par la poste, donc à la merci du facteur : le cadenas est appelé *clé publique*.
- Vous recevez le cadenas et l'utilisez pour fermer une boîte contenant votre lettre : le facteur ne peut pas l'ouvrir car il n'a pas la clé.
- Josette reçoit donc la boîte fermée : elle utilise sa clé, qu'elle seule possède, pour ouvrir la boîte et se pâmer devant vos élans épistolaires.

- Votre amie Josette veut recevoir de vous une lettre d'amour mais elle a peur que le facteur l'intercepte et la lise.
- Elle fabrique donc dans son petit atelier une clé, un cadenas et une boîte vide .
- Elle vous envoie le cadenas, ouvert mais sans la clé, par la poste, donc à la merci du facteur : le cadenas est appelé *clé publique*.
- Vous recevez le cadenas et l'utilisez pour fermer une boîte contenant votre lettre : le facteur ne peut pas l'ouvrir car il n'a pas la clé.
- Josette reçoit donc la boîte fermée : elle utilise sa clé, qu'elle seule possède, pour ouvrir la boîte et se pâmer devant vos élans épistolaires.

- Votre amie Josette veut recevoir de vous une lettre d'amour mais elle a peur que le facteur l'intercepte et la lise.
- Elle fabrique donc dans son petit atelier une clé, un cadenas et une boîte vide .
- Elle vous envoie le cadenas, ouvert mais sans la clé, par la poste, donc à la merci du facteur : le cadenas est appelé *clé publique*.
- Vous recevez le cadenas et l'utilisez pour fermer une boîte contenant votre lettre : le facteur ne peut pas l'ouvrir car il n'a pas la clé.
- Josette reçoit donc la boîte fermée : elle utilise sa clé, qu'elle seule possède, pour ouvrir la boîte et se pâmer devant vos élans épistolaires.

- Votre amie Josette veut recevoir de vous une lettre d'amour mais elle a peur que le facteur l'intercepte et la lise.
- Elle fabrique donc dans son petit atelier une clé, un cadenas et une boîte vide .
- Elle vous envoie le cadenas, ouvert mais sans la clé, par la poste, donc à la merci du facteur : le cadenas est appelé *clé publique*.
- Vous recevez le cadenas et l'utilisez pour fermer une boîte contenant votre lettre : le facteur ne peut pas l'ouvrir car il n'a pas la clé.
- Josette reçoit donc la boîte fermée : elle utilise sa clé, qu'elle seule possède, pour ouvrir la boîte et se pâmer devant vos élans épistolaires.

Le principe se ramène à :

- je fabrique une fonction π définie sur $\mathbb{N}$ qui possède une réciproque σ.
- On suppose qu'on peut fabriquer de telles fonctions mais que si l'on ne connaît que π, il est (quasiment) impossible de retrouver σ.
- La fonction π est donc la clé publique : vous envoyez $\pi(\text{message})$ à Josette.
- Celle-ci calcule $\sigma(\pi(\text{message})) = \text{message}$.
- Josette est la seule à pouvoir décoder car elle seule connaît σ.

Le principe se ramène à :

- je fabrique une fonction π définie sur $\mathbb{N}$ qui possède une réciproque σ.
- On suppose qu'on peut fabriquer de telles fonctions mais que si l'on ne connaît que π, il est (quasiment) impossible de retrouver σ.
- La fonction π est donc la clé publique : vous envoyez $\pi(\text{message})$ à Josette.
- Celle-ci calcule $\sigma(\pi(\text{message})) = \text{message}$.
- Josette est la seule à pouvoir décoder car elle seule connaît σ.

Le principe se ramène à :

- je fabrique une fonction π définie sur $\mathbb{N}$ qui possède une réciproque σ.
- On suppose qu'on peut fabriquer de telles fonctions mais que si l'on ne connaît que π, il est (quasiment) impossible de retrouver σ.
- La fonction π est donc la clé publique : vous envoyez $\pi(\text{message})$ à Josette.
- Celle-ci calcule $\sigma(\pi(\text{message})) = \text{message}$.
- Josette est la seule à pouvoir décoder car elle seule connaît σ.

Le principe se ramène à :

- je fabrique une fonction π définie sur $\mathbb{N}$ qui possède une réciproque σ.
- On suppose qu'on peut fabriquer de telles fonctions mais que si l'on ne connaît que π, il est (quasiment) impossible de retrouver σ.
- La fonction π est donc la clé publique : vous envoyez $\pi(\text{message})$ à Josette.
- Celle-ci calcule $\sigma(\pi(\text{message})) = \text{message}$.
- Josette est la seule à pouvoir décoder car elle seule connaît σ.

Le principe se ramène à :

- je fabrique une fonction π définie sur $\mathbb{N}$ qui possède une réciproque σ.
- On suppose qu'on peut fabriquer de telles fonctions mais que si l'on ne connaît que π, il est (quasiment) impossible de retrouver σ.
- La fonction π est donc la clé publique : vous envoyez $\pi(\text{message})$ à Josette.
- Celle-ci calcule $\sigma(\pi(\text{message})) = \text{message}$.
- Josette est la seule à pouvoir décoder car elle seule connaît σ.

Comment faire cela informatiquement ?...

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- On commence par fabriquer deux nombres premiers quelconques mais suffisamment grands, par exemple de l'ordre de 1000 bits (300 chiffres décimaux).
- Ces deux nombres p et q ne sont connus que d'une personne : c'est la *clé privée*.
- Josette forme le produit de ces deux nombres : $n = p \times q$.
- Ce nombre n est le module publique de Josette.
- Il reste à former le nombre e qui est la puissance de cryptage publique.
- Josette commence par former le nombre $\varphi_n = (p-1)(q-1)$.
- Josette doit choisir un nombre e premier avec φ_n. En pratique, elle choisit un grand nombre premier au hasard.
- Il reste à Josette à calculer sa puissance de décryptage secrète : elle l'obtient en cherchant l'inverse d de e modulo φ_n.

- Vous avez donc reçu (e, n), la clé publique de Josette. Vous découpez votre message en petits morceaux à l'aide d'un chiffrement par blocs.
- Vous calculez chaque bloc à la puissance e modulo n.
- Josette reçoit ce message et le décrypte grâce à sa clé secrète d qu'elle est la seule à connaître dans tout l'univers...

- Vous avez donc reçu (e, n), la clé publique de Josette. Vous découpez votre message en petits morceaux à l'aide d'un chiffrement par blocs.
- Vous calculez chaque bloc à la puissance e modulo n.
- Josette reçoit ce message et le décrypte grâce à sa clé secrète d qu'elle est la seule à connaître dans tout l'univers...

- Vous avez donc reçu (e, n), la clé publique de Josette. Vous découpez votre message en petits morceaux à l'aide d'un chiffrement par blocs.
- Vous calculez chaque bloc à la puissance e modulo n.
- Josette reçoit ce message et le décrypte grâce à sa clé secrète d qu'elle est la seule à connaître dans tout l'univers...