Mathématiques discrètes

Marc-André Désautels

Table des matières

Préface

Ce document est un livre Quarto.

Pour en apprendre davantage sur les livres Quarto, visitez https://quarto.org/docs/books.

1 Systèmes de numération positionnelle

Un système de numération est un ensemble de règles qui permettent de représenter des nombres. Le plus ancien est probablement le système unaire où le symbole | représente l'entier un, || représente l'entier deux, ||| pour trois, |||| pour quatre et ainsi de suite. Ce système atteint vite ses limites, mais il permet de mettre en évidence le fait qu'il existe plusieurs façons de représenter les entiers.

Nom français	**Système unaire**	**Système décimal**	**Chiffres romains**
Zéro		0	
Un	|	1	I
Deux	||	2	II
Trois	|||	3	III
⋮	⋮	⋮	⋮
Douze	|||| |||| ||||	12	XII
⋮	⋮	⋮	⋮

Dans la table ci-dessus, on remarque que sur une ligne donnée, on retrouve quatre manières différentes de représenter le même entier. Pour le reste de cette section, il sera important de dissocier la **représentation** d'un nombre et sa **valeur**.

Définition 1.1 (Système de numération)**.** Un **système de numération** permet de compter des objets et de les représenter par des nombres. Un système de numération **positionnel** possède trois éléments:

- Base b (un entier supérieur à 1)
- Symboles (digits): 0, 1, 2, ..., b-1
- Poids des symboles selon la position et la base, où poids=base$^{\text{position}}$

ℹ Note

Lorsque plusieurs bases interviennent dans un même contexte, on écrit $(a_n \dots a_1 a_0)_b$ pour indiquer que le nombre représenté en base b.

Définition 1.2 (Représentation polynomiale)**.** Le système positionnel utilise la **représentation polynomiale**. Celle-ci est donnée par:

$$
\begin{aligned}
(a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-m})_b = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0 + \dots \\
\dots + a_{-1} b^{-1} + \dots + a_{-m} b^{-m}
\end{aligned}
$$

où b est la **base** et les a_i sont des **coefficients** (les symboles de votre système de numération).

1.1 Système décimal

Il s'agit du système de numération le plus utilisé dans notre société. On peut le résumer avec les trois règles suivantes.

- Base = 10
- Symboles ordonnés qu'on nomme les *chiffres* : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Le poids des symboles est donné par 10^{position}

Ainsi, l'écriture "197 281" signifie:

$$197\ 281 = 1 \cdot 10^5 + 9 \cdot 10^4 + 7 \cdot 10^3 + 2 \cdot 10^2 + 8 \cdot 10^1 + 1 \cdot 10^0$$

Exemple 1.1. Représentez le nombre 3482 sous une forme de numération positionnelle.

1.2 Système binaire

Ce concept est essentiel en informatique, puisque les processeurs des ordinateurs sont composés de transistors ne gérant que deux états chacun (0 ou 1). Un calcul informatique n'est donc qu'une suite d'opérations sur des paquets de 0 et de 1, appelés **bits**.

- Base = 2
- Symboles ordonnés qu'on nomme les *bits*: 0, 1
- Le poids des symboles est donné par 2^{position}

! Important

En base 2, le *chiffre* 2 n'existe pas (c'est un **nombre**); tout comme le *chiffre* 10 n'existe pas en base 10 (c'est un **nombre**).

Nombres binaires en `Python`

Pour indiquer qu'un nombre est en binaire dans `Python`, il faut le faire précéder par `0b`. Pour convertir un nombre en binaire, on utilise la command `bin`.

Exemple 1.2. Quels sont les nombres qui, dans la base deux, succèdent à $(0)_2$?

```python
depart = 0b0
for i in range(6):
    depart = depart + 1
    print(bin(depart))
```

```
0b1
0b10
0b11
0b100
0b101
0b110
```

Exemple 1.3. Quels sont les nombres qui, dans la base deux, succèdent à $(1110)_2$?

```
depart = 0b1110
for i in range(6):
    depart = depart + 1
    print(bin(depart))
```

```
0b1111
0b10000
0b10001
0b10010
0b10011
0b10100
```

Exemple 1.4. Convertissez le nombre $(11001)_2$ en décimal.

Exemple 1.5. Convertissez les nombres suivants en décimal.

(a) $(110)_2 =$
(b) $(101101)_2 =$
(c) $(0{,}1011)_2 =$
(d) $(110{,}101)_2 =$

1.3 Système octal

Le système de numération octal est le système de numération de base 8, et utilise les chiffres de 0 à 7. D'après l'ouvrage de Donald Knuth's, *The Art of Computer Programming*, il fut inventé par le roi Charles XII de Suède.

- Base = 8
- Symboles ordonnés qu'on nomme les *chiffres*: 0, 1, 2, 3, 4, 5, 6, 7
- Le poids des symboles est donné par 8^{position}

Nombres octaux en `Python`

Pour indiquer qu'un nombre est en octal dans `Python`, il faut le faire précéder par `0o`. Pour convertir un nombre en octal, on utilise la commande `oct`.

Exemple 1.6. Quels sont les nombres qui, dans la base 8, succèdent à $(65)_8$?

```
depart = 0o65
for i in range(12):
    depart = depart + 1
    print(oct(depart))
```

```
0o66
0o67
0o70
0o71
0o72
0o73
0o74
```

```
0o75
0o76
0o77
0o100
0o101
```

1.4 Système hexadécimal

Le système hexadécimal est utilisé notamment en électronique numérique et en informatique car il est particulièrement commode et permet un compromis entre le code binaire des machines et une base de numération pratique à utiliser pour les ingénieurs. En effet, chaque chiffre hexadécimal correspond exactement à quatre chiffres binaires (ou bits), rendant les conversions très simples et fournissant une écriture plus compacte. L'hexadécimal a été utilisé la première fois en 1956 par les ingénieurs de l'ordinateur Bendix G-15.

- Base = 16
- Symboles ordonnés qu'on nomme les *chiffres*: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Le poids des symboles est donné par 16^{position}

On remarque qu'en base 16, les dix chiffres de 0 à 9 ne suffisent pas. Il faut donc se doter de 6 symboles additionnels. On utilise les lettres de A à F avec la signification suivante:

$$(A)_{16} = (10)_{10}, \quad (B)_{16} = (11)_{10}, \quad (C)_{16} = (12)_{10}$$

$$(D)_{16} = (13)_{10} \quad (E)_{16} = (14)_{10}, \quad (F)_{16} = (15)_{10}$$

Nombres hexadécimaux en `Python`

Pour indiquer qu'un nombre est en hexadécimal dans `Python`, il faut le faire précéder par `0x`.
Pour convertir un nombre en hexadécimal, on utilise la commande `hex`.

Exemple 1.7. Quels sont les nombres qui, dans la base 16, succèdent à $(\text{AAA})_{16}$?

```python
depart = 0xAAA
for i in range(12):
    depart = depart + 1
    print(hex(depart))
```

```
0xaab
0xaac
0xaad
0xaae
0xaaf
0xab0
0xab1
0xab2
0xab3
0xab4
0xab5
0xab6
```

Exemple 1.8. Trouvez la représentation en base 10 de:

a) $(\mathrm{AB0})_{16}$
b) $(214{,}\mathrm{EA})_{16}$

! Important

Pour convertir un nombre de la base b vers la base 10 (décimal), on trouve sa représentation polynomiale.

Conversion des nombres entiers vers décimal en `Python`

Pour convertir un nombre entier nb représenté dans la base b en `Python` en décimal, on utilise la commande `int(nb, b)`. Le nombre entier nb doit être représenté comme une chaîne de caractères (**string**).
Par exemple, si vous avez le nombre hexadécimal $A0F$, vous le convertissez de la manière suivante:

```python
int('A0F', 16)
```

```
2575
```

1.5 Division entière

Définition 1.3 (Divisibilité)**.** Si $a \in \mathbb{Z}$, $b \in \mathbb{Z}$ et $a \neq 0$, on dit que a **divise** b s'il existe un entier c tel que $b = ac$. L'entier a est alors appelé **facteur** de b.

Si a **divise** b, nous le notons $a \mid b$.

Théorème 1.1 (Divisibilité)**.** *Soit a, b et c des nombres entiers quelconques, avec $a \neq 0$.*

1. *Si $a \mid b$ et $a \mid c$ alors $a \mid (b+c)$ et $a \mid (b-c)$.*
2. *Si $a \mid b$ alors $a \mid (bc)$.*
3. *Si $a \mid b$ et $b \mid c$ alors $a \mid c$.*

Exemple 1.9. Vrai ou faux? Justifiez en invoquant une définition, un théorème, en donnant une preuve ou un contre-exemple.

a) $7 \mid 10$
b) $-5 \mid 10$
c) $100 \mid 10$
d) $5 \mid -10$

Théorème 1.2. *Soit a et d des entiers, avec $d > 0$. Il existe une seule paire d'entiers q et r satisfaisant*

$$0 \leq r < d \quad \text{et} \quad a = dq + r$$

Définition 1.4 (Diviseur, dividende, quotient, reste)**.** Considérons a et d des entiers, avec $d > 0$. Le Théorème 1.2 stipule qu'il existe une seule paire d'entiers q et r satisfaisant

$$a = dq + r \quad \text{et} \quad 0 \leq r < d$$

Par exemple, si $a = 17$ et $d = 3$, on a

$$17 = 3 \cdot 5 + 2 \quad \text{et} \quad 0 \leq 2 < 3$$

- L'entier $d = 3$ est appelé **diviseur**.
- L'entier $a = 17$ est appelé le **dividende**.
- L'entier $q = 5$ est appeléle **quotient** (notation: $q = a$ **div** d).
- L'entier $r = 2$ est appelé le **reste**.

1.6 Conversions de la base 10 vers une base b

Pour convertir un nombre entier de la base 10 vers une base b, il faut effectuer de façon successive des divisions en utilisant la Définition 1.4. Les restes des divisions successives correspondent aux coefficients de la représentation polynomiale (**lire de base en haut**).

1.6.1 Conversions vers binaire

Exemple 1.10. Convertissez les nombres suivants en binaire.

a) 115
b) 71

Nous pouvons utiliser la commande `bin` de `Python` pour convertir des **entiers** décimaux en binaire.

```
print(bin(115))
print(bin(71))
```

```
0b1110011
0b1000111
```

Pour convertir un nombre fractionnaire en binaire, il suffit de multiplier (plutôt que de diviser) la partie fractionnaire en notant les parties entières et fractionnaires obtenues. Il faut ensuite répéter ces étapes avec la nouvelle partie fractionnaire et poursuivre le processus jusqu'à ce que la partie fractionnaire soit nulle. Les parties entières des résultats de ces produits correspondent aux coefficients de la représentation polynomiale (**lire de haut en bas**).

Exemple 1.11. Convertissez les nombres suivants en binaire.

a) $(0{,}8125)_{10}$
b) $(0{,}15)_{10}$

> **!** Important
>
> La conversion en binaire ou en n'importe quelle base ne donne pas toujours une suite finie. Si c'est un nombre rationnel, la conversion donnera toujours une suite finie ou périodique.

Exemple 1.12. Convertissez en binaire les nombres suivants, en ne conservant que 6 chiffres pour la partie fractionnaire, au besoin.

a) $(51{,}375)_{10}$
b) $(564{,}32)_{10}$

1.6.2 Conversions vers octal

Nous pouvons utiliser la command `oct` de `Python` pour convertir des **entiers** décimaux en octal.

```
print(oct(115))
print(oct(71))
```

```
0o163
0o107
```

1.6.3 Conversions vers hexadécimal

Exemple 1.13. Convertissez les nombres décimaux suivants en hexadécimal.

a) $(176{,}47)_{10}$
b) $(69{,}28)_{10}$

Nous pouvons utiliser la command `hex` de `Python` pour convertir des **entiers** décimaux en hexadécimal.

```
print(hex(115))
print(hex(71))
```

```
0x73
0x47
```

1.6.4 Conversions binaire - hexadécimal

Une des raisons pour lesquelles le format hexadécimal a été inventé est qu'il est particulièrement simple de convertir un nombre binaire en nombre hexadécimal et inversement.

Hexa	0	1	2	3	4	5	6	7
Binaire	0000	0001	0010	0011	0100	0101	0110	0111
Hexa	8	9	A	B	C	D	E	F
Binaire	1000	1001	1010	1011	1100	1101	1110	1111

Pour convertir un nombre binaire, on regroupe par *paquets* de 4 chiffres à partir de la virgule (pour la partie entière et la partie fractionnaire).

Exemple 1.14. Convertissez les nombres binaires suivants en hexadécimal.

a) $(111001, 1101)_2$
b) $(1110001, 11\overline{001})_2$

Exemple 1.15. Convertissez les nombres hexadécimaux suivants en binaire.

a) $(537, 14)_{16}$
b) $(45B, 1\overline{DE})_{16}$

2 Représentation des nombres dans l'ordinateur

Lorsque nous voulons représenter des nombres dans un ordinateur, il faut distinguer deux cas biens différents; la représentation des nombres **entiers** et la représentation des nombres **fractionnaires**.

2.1 Représentation des entiers

En `Python`, contrairement à la plupart des langages informatiques, les entiers sont représentés avec une précision **infinie**. C'est-à-dire que la seule limite correspond à la mémoire interne de la machine que vous utilisez. Cependant, dans la majorité des langages informatiques, la précision de la représentation des entiers est **finie**, c'est-à-dire qu'un certain nombre de bits est alloué en mémoire pous stocker votre nombre et vous ne pouvez pas le dépasser.

Nous pouvons connaître le nombre de bits utilisés par `Python` dans la représentation d'un entier en utilisant la fonction `getsizeof` du module `sys`.

```
from sys import getsizeof

n1 = 2**32
n2 = 2**128
print(getsizeof(n1), getsizeof(n2))
```

```
32 44
```

Pour étudier le comportement d'entiers ayant une taille fixe, on peut utiliser le module `numpy`. Ce module possède plusieurs classes d'entiers à taille fixe.

2.1.1 Entiers non signés

Définition 2.1 (Entiers non signés (nombres positifs))**.** Un nombre **entier non signé** (positif) est représenté par un nombre de bits préalablement fixé. Au besoin, on complète le nombre par des zéros à gauche fin d'avoir le nombre total de bits choisi.

Les entiers non signés à taille fixe en `Python`

- `numpy.ubyte`: entier non signé sur 8 bits
- `numpy.ushort`: entier non signé sur 16 bits
- `numpy.uintc`: entier non signé sur 32 bits
- `numpy.uint`: entier non signé sur 64 bits

Exemple 2.1. Transformez les entiers décimaux suivants en entiers non signés sur un octet (huit bits).

a) 143

b) 15
c) 30

```
import numpy as np

print(bin(np.ubyte(143)), bin(np.ubyte(15)), bin(np.ubyte(30)))
```

```
0b10001111 0b1111 0b11110
```

Soyez prudents!

Si on tente d'écrire un nombre entier qui dépasse la capacité du format, nous n'obtenons pas nécessairement un message d'erreur, il faut donc être très prudents. Par exemple, le format `numpy.byte` peut représenter les entiers de 0 à 255. Si nous tentons de représenter 256, nous obtenons:

```
import numpy as np

print(np.uint8(256))
```

```
0
```

Ce genre d'erreur est appelée un dépassement d'entier. Un dépassement d'entier (*integer overflow*) est, en informatique, une condition qui se produit lorsqu'une opération mathématique produit une valeur numérique supérieure à celle représentable dans l'espace de stockage disponible. Par exemple, l'ajout d'une unité au plus grand nombre pouvant être représenté entraîne un dépassement d'entier.

Le dépassement d'entier le plus célèbre de ces dernières années est très probablement celui qui causa la destruction de la fusée Ariane 5, lors de son vol inaugural, le 4 juin 1996.

Exemple 2.2. Quel est le plus grand entier non signé pouvant être représenté avec:

a) 8 bits?
b) 32 bits?
c) n bits?

2.1.2 Entiers signés

Pour travailler avec des entiers qui peuvent être positifs ou négatifs, il faut inclure le signe du nombre dans sa représentation, et l'on parle alors d'entiers signés.

Définition 2.2 (Entiers signés (représentation signe et module))**.** Un nombre **entier signé** (généralement représenté dans un octet) est un nombre où le 1er bit (à gauche) est réservé au signe, et les autres bits permettent d'indiquer la valeur absolue du nombre. Pour indiquer qu'un nombre est positif (+), le 1er bit est `0`, et pour un nombre négatif (-), le 1er bit est `1`.

Les entiers signés à taille fixe en `Python`

- `numpy.byte`: entier signé sur 8 bits
- `numpy.short`: entier signé sur 16 bits

- `numpy.intc`: entier signé sur 32 bits
- `numpy.int_`: entier signé sur 64 bits

Exemple 2.3. Complétez les tableaux suivants qui indiquent la représentation signe et module sur 4 bits.

Base 2	Base 10
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Base 2	Base 10
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

En utilisant les nombres entiers signés:

- On peut écrire autant de nombres positifs que de négatifs.
- Pour un nombre exprimé avec n bits, les valeurs extrêmes sont $\pm(2^{n-1} - 1)$

Exemple 2.4. Quelles sont les valeurs extrêmes pour des entiers signés représentés sur 4 bits?

Inconvénients de la représentation signe et module

- Il y a deux zéros! Un *zéro* positif (0000 0000) et un *zéro* négatif (1000 0000).
- Les opérations arithmétiques ne se font pas de la même manière qu'habituellement. Par exemple, sur 4 bits:
  - **Base 2**: 0100 + 1011 = 1111
  - **Base 10**: +4 + -3 = -7! (**FAUX!**)

Exemple 2.5. Écrivez la représentation signe et module sur 8 bits de:

a) 15

a) -15

a) -10

a) Quel est l'intervalle de nombres entiers *signés* pouvant être représentés avec:

   i. 8 bits?
   ii. 16 bits?

2.2 Représentation des nombres en virgule flottante

La virgule flottante est une méthode d'écriture de nombres fréquemment utilisée dans les ordinateurs, équivalente à la notation scientifique en numération binaire. Elle consiste à représenter un nombre par :

- un signe (égal à -1 ou 1) ;
- une mantisse (aussi appelée significande) ;
- et un exposant (entier relatif, généralement borné).

Par exemple:

$$1,3254 = \underbrace{13254}_{\text{mantisse}} \times \overbrace{10^{-4}}^{\text{exposant}}$$

2.2.1 La norme IEEE754

En informatique, l'IEEE 754 est une norme sur l'arithmétique à virgule flottante mise au point par le *Institute of Electrical and Electronics Engineers*. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante avec les CPU et les FPU. La norme définit les formats de représentation des nombres à virgule flottante (signe, mantisse, exposant, nombres dénormalisés) et valeurs spéciales (infinis et NaN), en même temps qu'un ensemble d'opérations sur les nombres flottants. Il décrit aussi cinq modes d'arrondi et cinq exceptions (comprenant les conditions dans lesquelles une exception se produit, et ce qui se passe dans ce cas).

Format général

Un nombre flottant est formé de trois éléments : la mantisse, l'exposant et le signe. Le bit de poids fort est le bit de signe : si ce bit est à 1, le nombre est négatif, et s'il est à 0, le nombre est positif. Les e bits suivants représentent l'exposant biaisé (sauf valeur spéciale), et les m bits suivants (m bits de poids faible) représentent la mantisse.

Figure 2.1: Format général d'un nombre en virgule flottante.

L'exposant peut être positif ou négatif. Cependant, la représentation habituelle des nombres signés (complément à 2) rendrait la comparaison entre les nombres flottants un peu plus difficile. Pour régler ce problème, l'exposant est « biaisé », afin de le stocker sous forme d'un nombre non signé.

Ce biais est de 2e—1 − 1 (e représente le nombre de bits de l'exposant) ; il s'agit donc d'une valeur constante une fois que le nombre de bits e est fixé.

Nous remplaçons $\mathbb{R}$ par l'ensemble $\mathbb{F}$ des nombres à virgules flottantes, dont les membres sont zéro et tous les nombres de la forme:

$$\pm(1+f)\times 2^e$$

où $1+f$ représente la mantisse et où

$$\begin{aligned} f &= \sum_{i=1}^{m} a_i 2^{-i}, \qquad a_i \in \{0,1\} \\ &= a_1 2^{-1} + a_2 2^{-2} + \ldots + a_m 2^{-m} \end{aligned} \tag{2.1}$$

pour un entier fixé m, la taille de la mantisse. L'équation (Équation 2.1) représente des nombres dans l'intervalle $[1,2[$. De manière équivalente, nous pouvons écrire

$$f = 2^{-m}\sum_{i=1}^{m} a_i 2^{m-i} = 2^{-m} z, \qquad a_i \in \{0,1\} \tag{2.2}$$

pour un entier z dans l'ensemble $\{0,1,\ldots,2^m-1\}$.

Exemple 2.6. Écrivez toutes les mantisses possibles si le nombre de bits de la mantisse est de 1, c'est-à-dire $m=1$.

Exemple 2.7. Écrivez toutes les mantisses possibles si le nombre de bits de la mantisse est de 2, c'est-à-dire $m=2$.

Définition 2.3 (L'epsilon d'une machine)**.** L'epsilon d'une machine est défini comme le plus petit nombre qui, ajouté à un, donne un résultat différent de un.

En utilisant l'équation (Équation 2.1), nous remarquons que le plus petit nombre (autre que 0) possible est 2^{-m}.

Pour déterminer l'epsilon de la machine en `Python`, on utilise la commande `sys.float_info.epsilon` du module `sys`.

```python
import sys
sys.float_info.epsilon
```

```
2.220446049250313e-16
```

Nous pouvons aussi utiliser un petit programme pour déterminer l'epsilon de la machine.

```python
eps = 1.0
while eps + 1 > 1:
    eps /= 2
eps *= 2
print("L'epsilon machine est:", eps)
```

```
L'epsilon machine est: 2.220446049250313e-16
```

En clair, si nous additionnons un nombre plus petit que l'epsilon machine, le résultat reste inchangé.

```python
import sys
eps = sys.float_info.epsilon
print(1+eps, 1+eps/2)
```

```
1.0000000000000002 1.0
```

Float Toy

3 Logique

3.1 Logique propositionnelle

Définition 3.1 (Proposition)**.** Un énoncé qui est soit vrai, soit faux est appelé une **proposition**. La **valeur de vérité** d'une proposition est donc **VRAI** ou **FAUX**.

En `Python`, les valeurs de vérités sont données par `True` (**VRAI**) et `False` (**FAUX**).

Un énoncé qui n'est pas une proposition (comme un paradoxe, une phrase impérative ou interrogative) sera qualifié d'innaceptable.

Exemple 3.1. Les énoncés suivants sont des propositions:

- Les numéros de téléphones au Canada ont dix chiffres.
- La lune est faite de fromage.
- 42 est la réponse à la question portant sur la *vie, l'univers et tout ce qui existe.*
- Chaque nombre pair plus grand que 2 peut être exprimé comme la somme de deux nombres premiers.
- $3 + 7 = 12$

Les énoncés suivants ne sont **pas** des propositions:

- Voulez-vous du gâteau?
- La somme de deux carrés.
- $1 + 3 + 5 + 7 + ... + 2n + 1$.
- Va dans ta chambre!
- $3 + x = 12$

Nous utilisons une table de vérité pour montrer les valeurs de vérité de propositions composées.

3.1.1 La négation

Définition 3.2 (La négation)**.** Soit p une proposition. L'énoncé:

> *Il n'est pas vrai que p.*

est une autre proposition appelée **négation** de p, qui est représentée par $\neg p$. La proposition $\neg p$ se lit *non p*. La table de vérité de la négation est donnée ci-dessous.

p	$\neg p$

En `Python`, l'opérateur `not` permet de faire la négation d'une valeur de vérité.

```
def negation(p):
    return not p

print("p    non_p")
for p in [True, False]:
    non_p = negation(p)
    print(p, non_p)
```

```
p    non_p
True False
False True
```

3.1.2 La conjonction

Je suis une roche **ET** je suis une île.

Définition 3.3 (La conjonction)**.** Soit p et q deux propositions. La proposition p *et* q, notée $p \wedge q$, est vraie si à la fois p et q sont vraies. Elle est fausse dans tous les autres cas. Cette proposition est appelée la **conjonction** de p et de q. La table de vérité de la conjonction est donnée ci-dessous.

p	q	$p \wedge q$

En `Python`, l'opérateur `and` permet de faire la conjonction de deux valeurs de vérité.

```
def conjonction(p, q):
    return p and q

print("p    q    p_et_q")
for p in [True, False]:
    for q in [True, False]:
        p_et_q = conjonction(p, q)
        print(p, q, p_et_q)
```

```
p    q    p_et_q
True True True
True False False
False True False
False False False
```

3.1.3 La disjonction

Elle a étudié très fort **OU** elle est extrêmement brillante.

Définition 3.4 (La disjonction)**.** Soit p et q deux propositions. La proposition p *ou* q, notée $p \vee q$, est fausse si p et q sont fausses. Elle est vraie dans tous les autres cas. Cette proposition est appelée la **disjonction** de p et de q. La table de vérité de la disjonction est donnée ci-dessous.

p	q	$p \vee q$

En `Python`, l'opérateur `or` permet de faire la disjonction de deux valeurs de vérité.

```python
def disjonction(p, q):
    return p or q

print("p    q    p_ou_q")
for p in [True, False]:
    for q in [True, False]:
        p_ou_q = disjonction(p, q)
        print(p, q, p_ou_q)
```

```
p    q    p_ou_q
True True True
True False True
False True True
False False False
```

3.1.4 La disjonction exclusive

> Prenez **SOIT** deux Advil **OU** deux Tylenols.

Définition 3.5 (La disjonction exclusive)**.** Soit p et q deux propositions. La proposition p *ou exclusif* q, notée $p \oplus q$, est vraie si p et q ont des valeurs de vérité **différentes**. Elle est fausse dans tous les autres cas. Cette proposition est appelée la **disjonction exclusive** de p et de q. La table de vérité de la disjonction exclusive est donnée ci-dessous.

p	q	$p \oplus q$

En `Python`, il n'existe pas d'opérateur logique pour effectuer la disjonction exclusive. On peut par contre utiliser l'opérateur bit à bit `^` pour faire cette disjonction exclusive.

Exemple 3.2. Utilisez les opérateurs logiques vus précédemment pour construire la table de vérité de la disjonction exclusive dans `Python`.

```python
def disjonction_exclusive(p, q):
    return #REMPLACEZ MOI#

print("p    q    p_ou_exclusif_q")
for p in [True, False]:
    for q in [True, False]:
        p_ou_exclusif_q = disjonction_exclusive(p, q)
        print(p, q, p_ou_exclusif_q)
```

```
p    q    p_ou_exclusif_q
True True False
True False True
False True True
False False False
```

! Important

La disjonction exclusive signifie l'un ou l'autre, mais pas les deux.

3.1.5 L'implication

> **SI** vous avez 100 à l'examen final, **ALORS** vous obtiendrez A dans ce cours.

Définition 3.6 (L'implication)**.** Soit p et q deux propositions. L'**implication** $p \to q$ est une proposition qui est fausse quand p est vraie et que q est fausse, et qui est vraie dans tous les autres cas. Dans une implication, p est appelée l'**hypothèse** (ou l'**antécédent** ou la **prémisse**) et q, la **conclusion** (ou la **conséquence**). La table de vérité de l'implication' est donnée ci-dessous.

p	q	$p \to q$

En `Python`, il n'existe pas d'opérateur logique pour effectuer l'implication.

Exemple 3.3. Utilisez les opérateurs logiques vus précédemment pour construire la table de vérité de l'implication dans `Python`.

```python
def implication(p, q):
    return #REMPLACEZ MOI#

print("p    q    p_implique_q")
for p in [True, False]:
    for q in [True, False]:
        p_implique_q = implication(p, q)
        print(p, q, p_implique_q)
```

```
p    q    p_implique_q
True True True
True False False
False True True
False False True
```

> **!** Important
>
> Une implication peut être considérée comme un **contrat** qui échoue seulement si les conditions du contrat sont respectées mais les résultats ne sont pas remplis.

Comme les implications apparaissent constamment en mathématiques, il existe une vaste terminologie pour désigner $p \to q$. Voici les modes les plus courants:

- si p alors q;
- p implique q;
- p seulement si q;
- p est suffisant pour q;
- q si p;
- q chaque fois que p;
- q est nécessaire à p.

3.1.6 La biconditionnelle

Il pleut dehors **SI ET SEULEMENT SI** c'est un jour nuageux.

Définition 3.7 (La biconditionnelle)**.** Soit p et q deux propositions. La **biconditionnelle** $p \leftrightarrow q$ est une proposition qui est vraie quand p et q ont les mêmes valeurs de vérité et qui est fausse dans les autres cas. La table de vérité de la biconditionnelle est donnée ci-dessous.

p	q	$p \leftrightarrow q$

En `Python`, il n'existe pas d'opérateur logique pour effectuer la biconditionnelle.

Exemple 3.4. Utilisez les opérateurs logiques vus précédemment pour construire la table de vérité de la biconditionnelle dans `Python`.

```python
def biconditionnelle(p, q):
    return #REMPLACEZ MOI#

print("p    q    p_biconditionnelle_q")
for p in [True, False]:
    for q in [True, False]:
        p_biconditionnelle_q = biconditionnelle(p, q)
        print(p, q, p_biconditionnelle_q)
```

```
p    q    p_biconditionnelle_q
True True True
True False False
False True False
False False True
```

❗ Important

La biconditionnelle est vraie si les propositions ont la même valeur de vérité et fausse autrement.

Comme les biconditionnelles apparaissent constamment en mathématiques, il existe une vaste terminologie pour désigner $p \leftrightarrow q$. Voici les modes les plus courants:

- p si et seulement si q;
- p est nécessaire et suffisante pour q;
- si p alors q et réciproquement.

Définition 3.8 (Réciproque, contraposée et inverse)**.**

- La **réciproque** de la proposition $p \to q$ est la proposition $q \to p$.
- La **contraposée** de la proposition $p \to q$ est la proposition $\neg q \to \neg p$.
- L'**inverse** de la proposition $p \to q$ est la proposition $\neg p \to \neg q$.

3.2 Équivalences propositionnelles

Une proposition composée est une proposition formée de plusieurs connecteurs logiques.

Définition 3.9 (Tautologie, contradiction et contingence)**.** Une proposition composée qui est toujours vraie, quelle que soit la valeur de vérité des fonctions qui la compose est appelée une **tautologie**. Une proposition composée qui est toujours fausse est appelée une **contradiction**. Finalement, une proposition qui n'est ni une tautologie ni une contradiction est appelée une **contingence**.

Exemple 3.5. Remplissez la table de vérité suivante et dites si les propositions composées sont des tautologies, des contradictions ou des contingences.

p	q	$p \vee \neg p$	$p \wedge \neg p$

Exemple 3.6. Le code ci-dessous révèle la table de vérité de la proposition composée $(p \wedge q) \vee \neg q$.

```python
def conjonction(p, q):
    return p and q

def disjonction(p, q):
    return p or q

print("p    q    a")
for p in [True, False]:
```

```
    for q in [True, False]:
        a = disjonction(conjonction(p, q), not q)
        print(p, q, a)
```

```
p    q    a
True True True
True False True
False True False
False False True
```

De quelle manière pouvez-vous modifier le code précédent pour obtenir la table de vérité de la proposition composée $(p \vee \neg q) \wedge \neg p$?

Lorsque vous créez votre table de vérité, il est crucial que vous soyiez systématique pour vous assurer d'avoir toutes les valeurs de vérité possibles pour chacune des propositions simples. Chaque proposition a deux valeurs de vérité possibles, le nombre de lignes de la table devrait être égal à 2^n, où n est le nombre de propositions. Vous devriez également considérer de briser vos propositions complexes en plus petites propositions.

Exemple 3.7. L'extrait de code suivant fait intervenir les variables booléennes p, q et r. Chacune de ces variables peut prendre les valeurs **vrai** ou **faux**. Pour chaque bloc indiqué, donnez toutes les valeurs possibles pour p, q et r au moment où le bloc est atteint.

```
if (p and q):
    if r:
        #BLOC 1#
    else:
        #BLOC 2#
else:
    #BLOC 3#
```

p	q	r	
V	**V**	**V**	
V	**V**	**F**	
V	**F**	**V**	
V	**F**	**F**	
F	**V**	**V**	
F	**V**	**F**	
F	**F**	**V**	
F	**F**	**F**	

Définition 3.10 (Équivalences de propositions)**.** Les propositions p et q sont dites **logiquement équivalentes** si la proposition $p \leftrightarrow q$ est une tautologie. Ainsi, deux propositions sont logiquement équivalentes si elles ont la même table de vérité, c'est-à-dire la même valeur de vérité dans tous les cas possibles.

La notation $p \equiv q$ signifie que p et q sont équivalentes.

Exemple 3.8. Vérifiez l'équivalence suivante à l'aide d'une table de vérité.

$$p \to q \equiv \neg p \vee q$$

p	q	
V	**V**	
V	**F**	
F	**V**	
F	**F**	

Exemple 3.9. Vérifiez l'équivalence suivante à l'aide d'une table de vérité.

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

p	q	
V	**V**	
V	**F**	
F	**V**	
F	**F**	

Pour gagner du temps, on note les équivalences fréquemment utilisées dans une table et on leur donne un nom ou un numéro afin d'y faire référence.

Table 3.11: Équivalences logiques

Nom	**Équivalence 1**	**Équivalence 2**
Identité	$p \wedge \mathbf{V} \equiv p$	$p \vee \mathbf{F} \equiv p$
Domination	$p \vee \mathbf{V} \equiv \mathbf{V}$	$p \wedge \mathbf{F} \equiv \mathbf{F}$
Idempotence	$p \vee p \equiv p$	$p \wedge p \equiv p$
Double négation	$\neg(\neg p) \equiv p$	
Commutativité	$p \wedge q \equiv q \wedge p$	$p \vee q \equiv q \vee p$
Associativité	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distributivité	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Lois de De Morgan	$\neg(p \wedge q) \equiv \neg p \vee \neg q$	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
Absorption	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
Négation	$p \vee \neg p \equiv \mathbf{V}$	$p \wedge \neg p \equiv \mathbf{F}$

Table 3.12: Équivalences logiques (implications)

Numéro	**Implication**
1	$p \to q \equiv \neg p \vee q$
2	$p \to q \equiv \neg q \to \neg p$
3	$p \vee q \equiv \neg p \to q$
4	$p \wedge q \equiv \neg(p \to \neg q)$
5	$\neg(p \to q) \equiv p \wedge \neg q$
6	$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
7	$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
8	$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
9	$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

Table 3.13: Équivalences logiques (biconditionnelles)

Numéro	Biconditionnelle
1	$p \leftrightarrow q \equiv (p \to q) \wedge (q \to q)$
2	$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
3	$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
4	$p \leftrightarrow q \equiv \neg(p \wedge \neg q) \wedge \neg(\neg p \wedge q)$
5	$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

Exemple 3.10. Vérifiez que la proposition

$$\neg(p \to q) \to \neg q$$

est une tautologie

a. à l'aide d'une table de vérité;

p	q	
V	**V**	
V	**F**	
F	**V**	
F	**F**	

b. sans l'aide d'une table de vérité, en utilisant les tableaux d'équivalences.

! Propositions équivalentes ou non?

Pour démontrer que les propositions **ne sont pas** équivalentes, il suffit de fournir des valeurs de p, q et r pour lesquelles elles diffèrent. Pour démontrer que les propositions sont équivalentes, on peut procéder de l'une des trois façons suivantes.

1. Fournir leur table de vérité.
2. Utiliser la Table 3.11, la Table 3.12 ou la Table 3.13.
3. Formuler une explication en mots qui montre que les deux propositions sont vraies, ou encore que les deux sont fausses, exactement pour les mêmes combinaisons de valeur de vérité des variables propositionnelles.

3.3 Prédicats et quantificateurs

Un énoncé contenant une ou plusieurs variables tel que

$$x < 10 \quad \text{ou} \quad x + 2 = 7 - y$$

n'est pas une proposition pusique, tant que la valeur de x ou y n'est pas connue, on ne peut dire s'il est vrai ou faux.

Définition 3.11 (Terminologie)**.** Dans l'énoncé "$x < 10$", x est le **sujet**, et "est inférieur à 10" est le **prédicat**. Notons $P(x)$ l'énoncé $x < 10$. On dit que P est une **fonction propositionnelle**.

Une fonction propositionnelle $P(x)$ prend la valeur vrai ou faux quand x est précisé. Par exemple:

- $P(8)$ est une proposition vraie. On écrira parfois $P(8)$ est vrai (au masculin, en sous-entendant l'énoncé est vrai, ou même $P(8) \equiv \mathbf{V}$).
- $P(13)$ est une proposition fausse.
- $P(\text{Marc-André})$ n'est pas une proposition, car *Marc-André* n'est pas une valeur possible pour la variable x.

L'ensemble des valeurs possibles pour la variable x est appelé **univers du discours**, ou **domaine** de la fonction P.

Définition 3.12 (Quantificateurs)**.**

$$\forall: \text{ quantificateur universel} \qquad \exists: \text{ quantificateur existentiel}$$

$\forall\ x\ P(x)$: signifie “Pour toutes les valeurs de x dans l'univers du discours, $P(x)$”. Ou encore “Quel que soit x (dans l'univers du discours), $P(x)$”.

$\exists\ x\ P(x)$: signifie “Il existe un élément de x dans l'univers du discours tel que $P(x)$”. Ou encore “Il y a au moins un x (dans l'univers du discours) tel que $P(x)$”. Ou encore “Pour un certain x (dans l'univers du discours), $P(x)$”.

Notation. Certains auteurs mettent une virgule avant la fonction propositionnelle, surtout quand celle-ci est composée. Par exemple: $\forall\ x,\ (P_1(x) \to P_2(x) \vee P_3(x))$. Par ailleurs, si l'ensemble U n'a pas déjà été identifié, on peut préciser que la variable x prendra des valeurs dans l'ensemble U ainsi: $\exists\ x \in U,\ P(x)$.

Lorsque l'univers du discours est un ensemble fini $\{x_1, x_2, \dots, x_n\}$, on a les équivalences logiques suivantes:

$$\forall\ x\ P(x) \equiv P(x_1) \wedge P(x_2) \wedge \cdots \wedge P(x_n)$$
$$\exists\ x\ P(x) \equiv P(x_1) \vee P(x_2) \vee \cdots \vee P(x_n)$$

La quantification universelle $\forall\ x\ P(x)$ est vraie quand $P(x)$ est vraie pour toutes les valeurs de x dans l'univers du discours U. Elle est donc fausse s'il existe un x de U pour lequel $P(x)$ est fausse. Un tel élément est appelé un **contre-exemple** de $\forall\ x\ P(x)$.

La quantification existentielle $\exists\ x\ P(x)$ est vraie s'il existe au moins une valeur x dans l'univers du discours telle que $P(x)$ est vraie. Elle est fausse si $P(x)$ est fausse pour toutes les valeurs possibles de x.

Ainsi, pour prouver un énoncé de la forme $\forall\ x\ P(x)$ est vrai, fournir un exemple de x tel que $P(x)$ est vrai ne suffit pas. Il faut montrer que la proposition $P(x)$ est vraie pour toutes les valeurs de x, ce qui peut s'avérer particulièrement **difficile lorsque U est un ensemble infini**. Il en va de même losqu'on veut prouver qu'un énoncé de la forme $\exists\ x\ P(x)$ est faux.

Table 3.15: Comment prouver qu'un énoncé quantifié est vrai ou faux quand l'univers du discours U est infini.

Pour prouver que	est **vrai**	est **faux**
$\exists\, x\, P(x)$	il suffit de fournir **un exemple**: un x de U tel que $P(x)$ est vrai.	il faut fournir un argument général pour montrer que $P(x)$ est faux quel que soit x de U.
$\forall\, x\, P(x)$	il faut fournir un argument général pour montrer que $P(x) est vrai quel que soit x de U.	il suffit de fournir un **contre-exemple**: un x de U tel que $P(x)$ est faux.

Exemple 3.11. Si l'univers du discours est l'ensemble des nombres réels et

$$\begin{aligned} &P(x) \text{ désigne } x \geq 0 \\ &Q(x) \text{ désigne } x \text{ est un nombre premier} \\ &R(x) \text{ désigne } 3^x + 4^x = 5^x \\ &S(x) \text{ désigne } x \geq 100 \end{aligned}$$

dites si chacun des énoncés suivants est une proposition vraie, une proposition fausse ou n'est pas une proposition. Donnez un exemple ou un contre-exemple le cas échéant. Dans le cas contraire, indiquez qu'un argument général est requis.

a. $\forall\ x\ P(x)$
b. $\forall\ x\ \neg P(x)$
c. $\forall\ x\ P(x^2)$
d. $\exists\ x\ P(x)$
e. $\exists\ x\ \neg P(x)$
f. $\exists\ x\ Q(x)$
g. $\exists\ x\ Q(x^2)$
h. $\forall\ x\ R(x)$
i. $P(x)$
j. $\forall\ x\ (S(x) \to P(x))$
k. $(\forall\ x\ P(x)) \to (\forall\ x\ S(x))$
l. $\forall\ x\ S(x+100)$
m. $\forall\ x\ S(x^2+100)$

Théorème 3.1 (Lois de De Morgan pour les quantificateurs)**.**

$$\neg\exists\ x\ P(x) \equiv \forall\ x\ \neg P(x) \qquad \neg\forall\ x\ P(x) \equiv \exists\ x\ \neg P(x)$$

Exemple 3.12. Si l'univers du discours est l'ensemble des étudiants du programme Sciences Informatique et Mathématique (ScIM) et $M(x)$ désigne l'énoncé *l'étudiant x peut modifier les fichiers du répertoire U*, traduisez clairement les propositions suivantes à l'aide des quantificateurs.

a. Tous les étudiants de ScIM peuvent modifier les fichiers du répertoire U.
b. Il est faux que tous les étudiants de ScIM peuvent modifier les fichiers du répertoire U.
c. Au moins un étudiant de ScIM peut modifier les fichiers du répertoire U.
d. Il est faux qu'au moins un étudiant de ScIM peut modifier les fichiers du répertoire U.
e. Aucun étudiant de ScIM ne peut modifier les fichiers du répertoire U.
f. Au moins un étudiant de ScIM ne peut pas modifier les fichiers du répertoire U.

De plus, déterminez les propositions ci-dessus qui sont équivalentes.

Exemple 3.13. Si l'univers du discours est l'ensemble des billes contenues dans un bol, et si

- $G(x)$ désigne *la bille x est grosse*
- $J(x)$ désigne *la bille x est jaune*
- $R(x)$ désigne *la bille x est rouge*
- $B(x)$ désigne *la bille x est bleue*

traduisez clairement les propositions suivantes en prenant soin de bien formuler les phrases.

a. $\forall\ x\ (R(x) \vee J(x))$
b. $(\forall\ x\ R(x)) \vee (\forall\ x\ J(x))$
c. Les propositions a. et b. sont-elles équivalentes?
d. $\exists\ x\ B(x)$
e. $\neg(\exists\ x\ B(x))$
f. Utilisez le quantificateur universel $\forall$ pour écrire une proposition équivalente à la précédente.
g. $\neg(\forall\ x\ R(x))$
h. Utilisez le quantificateur existentiel $\exists$ pour écrire une proposition équivalente à la précédente.
i. $\forall\ x\ (G(x) \rightarrow B(x))$
j. $\exists\ x\ (G(x) \wedge B(x))$
k. $(\exists\ x\ G(x)) \wedge (\exists\ x\ B(x))$
l. Les deux propositions précédentes sont-elles équivalentes?
m. Les deux propositions suivantes sont-elles équivalentes?

$$(\exists\ x\ R(x)) \vee (\exists\ x\ J(x)) \qquad \text{et} \qquad \exists\ x\ (R(x) \vee J(x))$$

n. Les deux propositions suivantes sont-elles équivalentes?

$$(\forall\ x\ R(x)) \wedge (\forall\ x\ G(x)) \qquad \text{et} \qquad \forall\ x\ (R(x) \wedge G(x))$$

3.4 Opérations bit à bit

3.5 Problèmes de logique

Les problèmes suivants se déroulent sur une île imaginaire où tous les habitants sont soit des **chevaliers**, qui disent toujours la vérité, soit des **fripons**, qui mentent toujours. Ces énigmes implique un visiteur qui rencontre un petit groupe d'habitants de l'île. La plupart du temps, le but du visiteur est de *déduire* les types des habitants à partir de leurs énoncés.

Voici un exemple type de problème possible.

Déduisez!

En vous promenant sur l'île, vous rencontrez trois habitants gardant un pont. Pour passer, vous devez déduire le type de chaque habitant. Chaque individu dit un seul énoncé:

- Individu A: Si je suis un fripon, alors il y a exactement deux chevaliers ici.
- Individu B: L'individu A ment.
- Individu C: Soit nous sommes tous des fripons ou alors au moins l'un d'entre nous est un chevalier.

Quels sont les types des trois individus?

Stratégies

Voici quelques stratégies que vous pouvez utiliser pour résoudre ce genre de problème:

- Commencez en supposant qu'un individu est d'un certain type. Soyez stratégique avec votre supposition, tentez de résoudre un énoncé **ET**.
  - Si un individu dit **ET**, supposez qu'il est un chevalier;
  - Si un individu dit **OU**, supposez qu'il est un fripon;
  - Si un individu dit **SI/ALORS**, supposez qu'il est un fripon;
  - Si un individu dit **SI ET SEULEMENT SI**, attendez de connaître la valeur de vérité de leur énoncé avant de faire une supposition.
- Lorsqu'un individu est un chevalier, vous pouvez continuer leur énoncé.
- Lorsqu'un individu est un fripon, vous pouvez continuer la négation de leur énoncé.
  - Partie 1 **ET** Partie 2 → **NON** Partie 1 **OU NON** Partie 2
  - Partie 1 **OU** Partie 2 → **NON** Partie 1 **ET NON** Partie 2
  - **SI** Partie 1, alors Partie 2 → Partie 1 **ET NON** Partie 2
- Soyez prudents avec les *si et seulement si*
  - Lorsqu'un *si et seulement si* est `VRAI`, alors les deux parties ont la **même** valeur de vérité.
  - Lorsqu'un *si et seulement si* est `FAUX`, alors les deux parties ont des valeurs de vérités **différentes**.
- Lorsque vous avez prouvé l'identité d'un individu, vous pouvez utiliser cette information partout dans le reste de l'énigme.
- Si vous avez suffisament d'information pour confirmer que l'énoncé d'un individu est `VRAI`, alors ils doivent être un chevalier.
- Si vous avez suffisament d'information pour confirmer que l'énoncé d'un individu est `FAUX`, alors ils doivent être un fripon.

3.5.1 Trois énoncés différents

Nous pouvons, dans la plupart des problèmes, regrouper les énoncés des habitants de l'île en trois formes distinctes.

3.5.1.1 Accusations et affirmations

Dans une accusation, un habitant A dit par exemple *B est un fripon* ou un énoncé équivalent comme *B ment toujours*. Dans une affirmation, l'habitant A dit par exemple *B est un chevalier* ou alors *B dit toujours la vérité*.

Exemple 3.14. Que pouvez-vous conclure si A et B sont reliés par une **accusation**?

Exemple 3.15. Que pouvez-vous conclure si A et B sont reliés par une **affirmation**?

3.5.1.2 Conjonctions de fripons

Un exemple de conjonction de fripons est lorsque A dit que B *est un chevalier ou je suis un fripon*, ou alors C *est un fripon et je suis un fripon*

Exemple 3.16. Que pouvez-vous conclure si A et B sont reliés par **ou je suis un fripon**?

Exemple 3.17. Que pouvez-vous conclure si A et B sont reliés par **et je suis un fripon**?

3.5.1.3 Énoncés de différences ou de similarités

Parfois un habitant A dira B *est de mon type* ou peut-être C *n'est pas de mon type.*

Exemple 3.18. Que pouvez-vous conclure si A dit que B *est de son type*?

Exemple 3.19. Que pouvez-vous conclure si A dit que C *n'est pas de son type*?

Il est intéressant de comparer ces énoncés avec ceux d'accusations et d'affirmations. Ces deux types d'énoncés sont réciproques en quelque sorte. Lorsqu'un habitant dit directement de quel type est un autre habitant (dans une accusation ou une affirmation), tout ce qu'on apprend c'est que la source et la cible sont similaires ou différents, sans apprendre leur type. Par contre, lorsqu'un habitant dit un énoncé par rapport aux similitudes ou aux différences, nous apprenons exactement de quel type la cible est, sans apprendre si elle est similaire ou différente de la source.

Exemple 3.20. Vous rencontrez trois habitants de l'île.

- A dit: B ne ment jamais.
- A dit: C est un chevalier ou je suis un fripon.

Exemple 3.21. Vous rencontrez trois habitants de l'île.

- A dit: B ment toujours.
- B dit: A n'est pas de mon type.

4 Théorie des ensembles

4.1 Notions de base sur les ensembles

Définition 4.1 (Ensemble, élément)**.** Un **ensemble** est un collection non ordonnée d'objets. Les objets sont appelés **éléments** de l'ensemble et on dit qu'ils appartiennent à l'ensemble.

Notation : $x \in F$ signifie que x est un élément de l'ensemble F. On dit aussi que x appartient à l'ensemble F.

Définition 4.2 (Ensemble fini ou infini, cardinalité)**.** Soit A un ensemble composé de n éléments distincts. On dit que A est un **ensemble fini** de **cardinalité** n et on note $|A| = n$. Un ensemble est dit **infini** s'il n'est pas fini.

Exemple 4.1. Soit l'ensemble $F = \{2, \pi, 7\}$. Utilisez les symboles introduits pour traduire les énoncés suivants: l'ensemble F contient 3 éléments, π appartient à F, 5 n'appartient pas à F.

On peut décrire un ensemble **en extension** (on énumère ses éléments que l'on place entre accolades)

$$A = \{5, 7, 9, 11\} \qquad B = \{1, 8, 27, 64\}$$

ou en **compréhension**, comme ceci:

$$A = \{x \in \mathbb{N} \mid (x \text{ est impair}) \wedge (5 \leq x \leq 11)\} \qquad B = \{x \in \mathbb{N} \mid (x \leq 64) \wedge (\exists\, y \in \mathbb{N},\ y^3 = x)\}$$

Pour créer un ensemble dans `Python`, nous allons utiliser une paire d'accolades `{ }` et placer les différents éléments de notre ensemble entre ces accolades en les séparant avec une virgule. De plus, nous pouvons vérifier si un élément appartient à l'ensemble en utilisant la commande `in`.

```
A={-2,0,1,4}
print(A, 1 in A, 5 in A)
```

```
{0, 1, 4, -2} True False
```

Pour calculer la cardinalité d'un ensemble dans `Python`, vous utilisez la fonction `len()`. En `Python`, il faut être prudent si on souhaite utiliser l'ensemble vide, $\emptyset$. Si vous utilisez `{}` pour décrire l'ensemble vide, `Python` va plutôt l'interpréter comme un *dictionnaire* vide. Vous devez plutôt utiliser la fonction `set()`.

```
A = {2,3,5,8}
B = set()
C = {0}
print(len(A), len(B), len(C))
```

```
4 0 1
```

4.2 Ensembles de nombres $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$

Nous détaillerons dans la Table 4.1, les ensembles de nombres les plus communs.

Table 4.1: Ensembles de nombres usuels.

Ensemble	**Description**
$\emptyset = \{\ \}$	Ensemble vide (ne contient aucun élément $\mid \emptyset \mid = 0$)
$\mathbb{N} = \{0, 1, 2, 3, ...\}$	Ensemble des nombres naturels
$\mathbb{N}^* = \{1, 2, 3, ...\}$	Ensemble des nombres naturels strictement positifs
$\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$	Ensemble des nombres entiers
$\mathbb{Z}^* = \{..., -2, -1, 1, 2, ...\}$	Ensemble des entiers non nuls
$\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{Z} \text{ et } q \neq 0\}$	Ensemble des nombres rationnels
$\mathbb{R}$	Ensemble des nombres réels
$\mathbb{R}^+ = \{x \in \mathbb{R} \mid x \geq 0\}$	Ensemble des nombres réels positifs
$\mathbb{C} = \{a + bi \mid a \in \mathbb{R} \text{ et } b \in \mathbb{R}\}$ avec $i^2 = -1$	Ensemble des nombres complexes

Exemple 4.2. Établissez un lien entre les ensembles décrits par compréhension aux parties a. à f. avec le même ensemble décrit par extension aux parties 1 à 6.

a. $\{x \in \mathbb{Z} \mid x^2 = 1\}$
b. $\{x \in \mathbb{Z} \mid x^3 = 1\}$
c. $\{x \in \mathbb{Z} \mid |x| \leq 2\}$
d. $\{x \in \mathbb{Z} \mid x^2 \leq 4\}$
e. $\{x \in \mathbb{Z} \mid x < |x|\}$
f. $\{x \in \mathbb{Z} \mid (x+1)^2 = x^2 + 2x + 1\}$

1. $\{-1, 0, 1\}$
2. $\{..., -3, -2, -1, 0, 1, 2, 3, ...\}$
3. $\{1\}$
4. $\{..., -3, -2, -1\}$
5. $\{-1, 1\}$
6. $\{-2, -1, 0, 1, 2\}$

> **Note**
>
> Lorsqu'il y a trop d'éléments dans un ensemble pour être en mesure de tous les écrire, nous utilisons souvent les trois-points (...) lorsque la suite d'éléments est claire. Par exemple, nous avons:
> $$\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$$

En `Python`, si vous avez un ensemble décrit par compréhension, il est particulièrement facile de le créer avec une `compréhension de liste`. L'idée est simple: simplifier le code pour le rendre plus lisible et donc plus rapide à écrire et plus simple à maintenir. La syntaxe est la suivante:

```
new_list = [function(item) for item in list if condition(item)]
new_list = {function(item) for item in list if condition(item)}
```

Par exemple, si vous voulez créer l'ensemble $\{x^3 \mid 0 \leq x < 10\}$, nous pouvons le faire en `Python` de la manière suivante:

```python
ensemble = {x**3 for x in range(10)}
liste = [x**3 for x in range(10)]
print(ensemble, liste)
```

```
{0, 1, 64, 512, 8, 343, 216, 729, 27, 125} [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

> **Note**
>
> Remarquez que dans l'ensemble, les éléments ne sont pas ordonnés, tandis qu'ils le sont dans la liste.

Définition 4.3 (Égalité d'ensembles)**.** Deux ensembles sont dits égaux si et seulement s'ils contiennent exactement les mêmes éléments.

$$A = B \leftrightarrow \forall\ x\ (x \in A \leftrightarrow x \in B)$$

Exemple 4.3. Les ensembles suivants sont-ils égaux?

$$\{1, 3, 5\} \stackrel{?}{=} \{3, 5, 1\}$$
$$\{1, 3, 5\} \stackrel{?}{=} \{\{1\}, \{3\}, \{5\}\}$$

Définition 4.4 (Sous-ensemble)**.** L'ensemble A est **sous-ensemble** de l'ensemble B si et seulement si tous les éléments de A sont aussi des éléments de B:

$$A \subseteq B \leftrightarrow \forall\ x\ (x \in A \rightarrow x \in B)$$

L'ensemble A est **sous-ensemble strict (ou propre)** de l'ensemble B si et seulement si tous les éléments de A sont aussi des éléments de B et A n'est pas égal à B:

$$A \subset B \leftrightarrow A \subseteq B \wedge\ A \neq B$$

Exemple 4.4. Convainquez-vous des affirmations suivantes.

$$\{1, 2\} \subseteq \{1, 2, 3, 4, 5\}$$
$$\{1, 2\} \subset \{1, 2, 3, 4, 5\}$$
$$\{2k \mid k \in \mathbb{N}\} = \{0, 2, 4, 6, ...\} \subset \mathbb{N}$$

Table 4.2: Notation de la théorie des ensembles.

Notation	Description
$\in$	$2 \in \{1, 2, 3\}$ indique que 2 est **un élément de** l'ensemble $\{1, 2, 3\}$.
$\notin$	$4 \notin \{1, 2, 3\}$ indique que 4 **n'est pas un élément de** l'ensemble $\{1, 2, 3\}$.
$\subseteq$	$A \subseteq B$ indique que A est un **sous-ensemble** de B: chaque élément de A est aussi un élément de B.
$\subset$	$A \subset B$ indique que A est un **sous-ensemble propre** de B: chaque élément de A est aussi un élément de B, mais $A \neq B$.

Théorème 4.1. *Pour tout ensemble A, on a :*

1. $\emptyset \subseteq A$

2. $A \subseteq A$

Théorème 4.2. *$A = B$ si et seulement si $A \subseteq B$ et $B \subseteq A$.*

En `Python`, nous pouvons utiliser la fonction `issubset` pour vérifier qu'un ensemble est sous-ensemble d'un autre.

```
A = {2,4,6,8,10,12}
B = {4,8,12}
print(A.issubset(B), B.issubset(A))
```

```
False True
```

4.3 Produit cartésien

Définition 4.5 (Produit cartésien)**.** Le **produit cartésien** des ennsembles A et B, noté $A \times B$, est L'ensemble de tous les couples (paires ordonnées) dont le premier élément appartient à A et le second, à B:

$$A \times B = \{(a, b) \mid \ a \in A \text{ et } b \in B\}$$

On généralise cette définition au produit cartésien de n ensembles:

$$A_1 \times A_2 \times \dots \times A_n = \{(a_1, a_2, \dots, a_n) \mid \ a_1 \in A_1, \dots, a_n \in A_n\}$$

Exemple 4.5. Décrivez en extension les produits cartésiens $A \times B$ et $B \times A$, où $A = \{0, 1, 2\}$ et $B = \{a, c\}$.

Définition 4.6 (Relation)**.** Une **relation** entre les ensembles A et B est un sous-ensemble du produit cartésien $A \times B$.

Exemple 4.6. Soit \$A={0,1,2} et \$B={a,c}. L'ensemble

$$R = \{(0, a), (1, c), (2, a)\} \subseteq A \times B$$

est une relation de A dans B.

Définition 4.7. L'**ensmeble des parties** de A, noté $\mathcal{P}(A)$, est l'ensemble de tous les sous-ensembles de A.

$$B \in \mathcal{P}(A) \leftrightarrow B \subseteq A$$

Exemple 4.7. Décrivez $\mathcal{P}(A)$, l'ensemble des parties de A, où $A = \{0, 1, 2\}$.

k	**Sous-ensembles de A ayant k éléments**	**Nombre de sous-ensembles**
0	$\emptyset$	1
1	$\{0\}$, $\{1\}$, $\{2\}$	3
2	$\{0, 1\}$, $\{0, 2\}$, $\{1, 2\}$	3
3	$\{0, 1, 2\}$	1

Exemple 4.8. Décrivez $\mathcal{P}(A)$, l'ensemble des parties de A, où $A = \{0, 1, 2, 3\}$.

k	Sous-ensembles de A ayant k éléments	Nombre de sous-ensembles
0		
1		
2		
3		
4		

4.4 Opérations sur les ensembles $\cap$, $\cup$, $\oplus$, $-$

Soit U l'ensemble universel et A et B des sous-ensembles de U. Les opérations suivantes génèrent des sous-ensembles de U.

Table 4.5: Les diverses opérations sur les ensembles.

Opération	Forme mathématique
Union	$\{x \in U \mid \; x \in A \;\vee\; x \in B\}$
Intersection	$\{x \in U \mid \; x \in A \;\wedge\; x \in B\}$
Différence	$\{x \in U \mid \; x \in A \;\wedge\; x \notin B\} = A \quad B$
Différence symétrique	$\{x \in U \mid \; x \in A \;\oplus\; x \in B\}$
Complément	$\{x \in U \mid \; x \notin A\} = U - A$

Vous pouvez effectuer ces opérations dans `Python` à l'aide des commandes suivantes:

Table 4.6: Les opérations sur les ensembles dans `Python`.

Opération	Commande `Python`
Union	`union`
Intersection	`intersection`
Différence	`difference`

```
A = {-3,-1,2,5}
B = {-1, 0, 2}
print(A.union(B))
```

```
{0, 2, 5, -3, -1}
```

```
A = {-3,-1,2,5}
B = {-1, 0, 2}
print(A.intersection(B))
```

```
{2, -1}
```

```
A = {-3,-1,2,5}
B = {-1, 0, 2}
print(A.difference(B))
```

```
{5, -3}
```

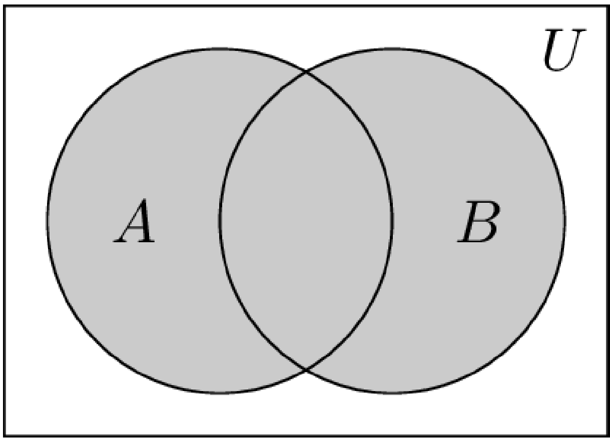

Figure 4.1: Union

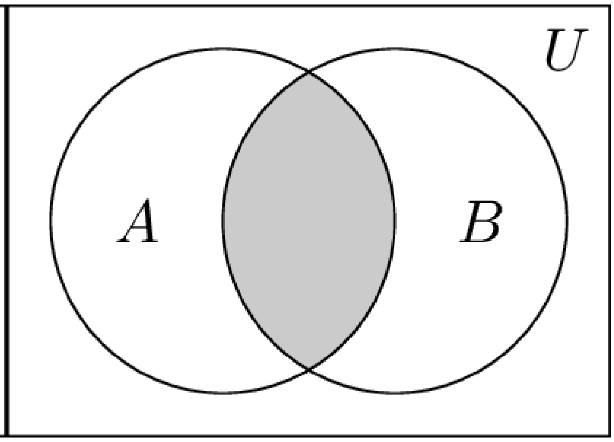

Figure 4.2: Intersection

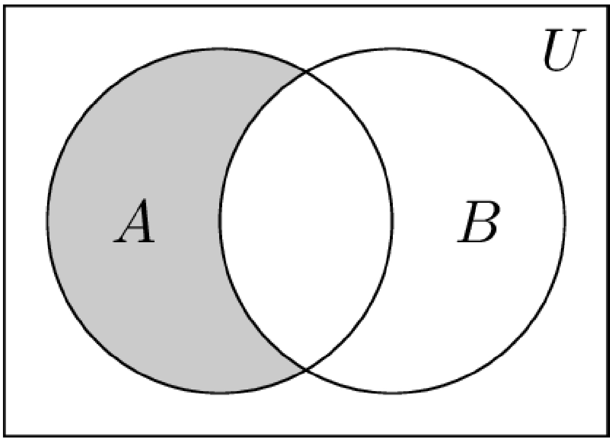

Figure 4.3: Différence

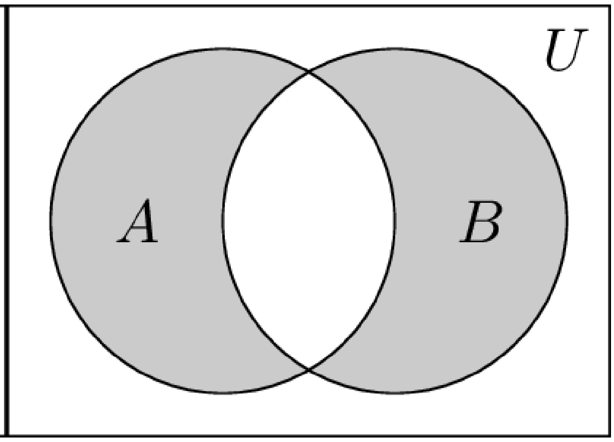

Figure 4.4: Différence symétrique

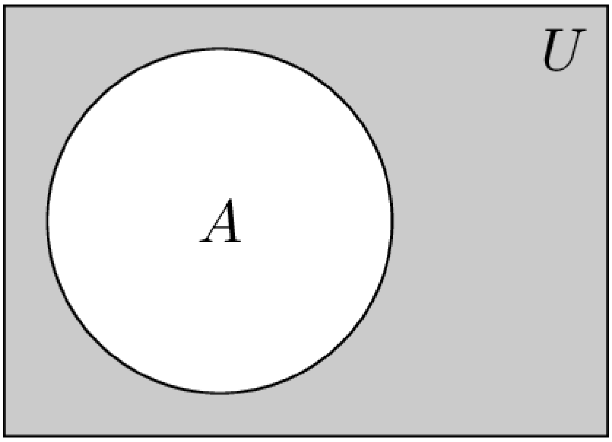

Figure 4.5: Complément

4.5 Représentation de sous-ensembles par trains de bits

4.6 Polygones convexes avec des opérations sur les ensembles

5 Fonctions

Définition 5.1 (Fonction)**.** Une **fonction** f d'un ensemble A vers un ensemble B est une règle qui, à chaque élément a de l'ensemble A, associe un et un seul élément b de l'ensemble B. Cet élément b est noté $f(a)$. On écrit parfois $(a, b) \in f$.

La notation usuelle pour désigner une fonction f d'un ensemble A vers un ensemble B est

$$f : A \to B$$

L'ensemble A est appelé le **domaine** de la fonction f, noté $\mathbf{dom}(f)$, et le sous-ensemble B formé des éléments atteints par f est appelé l'**image** de f, noté $\mathbf{ima}(f)$.

$$\mathbf{ima}(f) = \{b \in B \ \mid \ \exists a \in A,\ f(a) = b\} \subseteq B$$

Par ailleurs, on peut aussi voir une fonction f de A vers B comme un sous-ensemble du produit cartésien $A \times B$ ayant la propriété suivante:

$$\forall\ a \in A,\ \exists! b \in B,\ (a, b) \in f$$

où le symbole $\exists!$ désigne **il existe un et un seul**.

Exemple 5.1. Considérons T_8, l'ensemble des trains de bits de longueur 8 et la fonction $f : T_8 \to \mathbb{N}$ définie par

$$f(t) = \text{nombre de 0 dans le train de bits } t$$

Par exemple, $f(1100\ 1011) = 3$. Donnez le domaine et l'image de la fonction f.

5.1 Fonctions plancher et plafond

Définition 5.2 (Fonctions plancher et plafond)**.** La fonction **plancher** associe à tout nombre réel x, le plus grand entier n tel que $n \leq x$. On note $\lfloor x \rfloor = n$. La fonction **plafond** associe à tout nombre réel x, le plus petit entier n tel que $n \geq x$. On note $\lceil x \rceil = n$.

Exemple 5.2. Calculez les fonctions suivantes:

$$\left\lfloor \frac{1}{3} \right\rfloor =$$
$$\left\lceil \frac{1}{3} \right\rceil =$$
$$\lfloor -9,2 \rfloor =$$
$$\lceil -9,2 \rceil =$$

Théorème 5.1 (Propriétés des fonctions plancher et plafond)**.**

1. $\lfloor x \rfloor = n \leftrightarrow n \leq x < n + 1$
2. $\lceil x \rceil = n \leftrightarrow n - 1 < x \leq n$

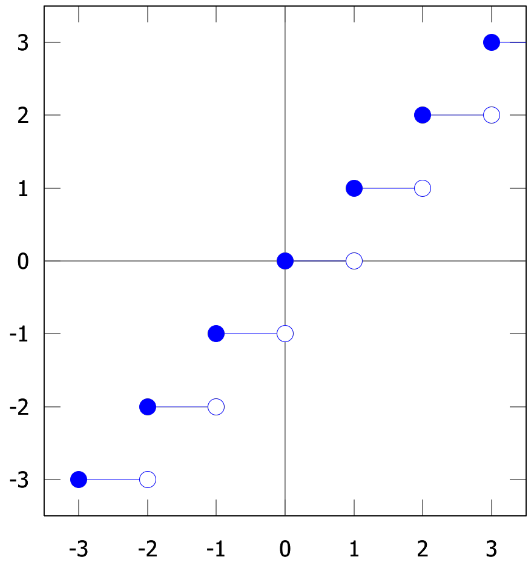

Figure 5.1: Fonction plancher

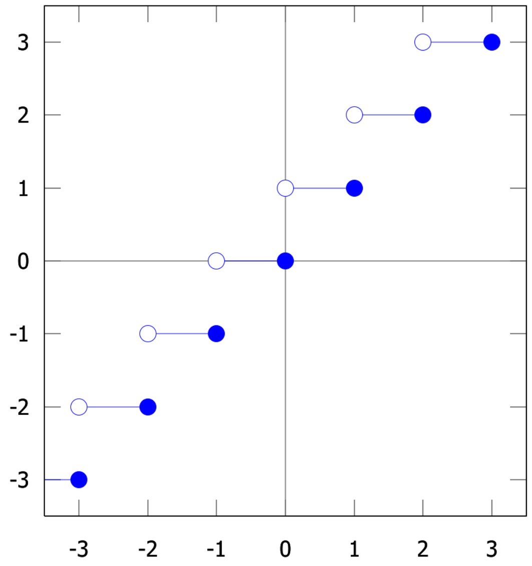

Figure 5.2: Fonction plafond

Figure 5.3: Les fonctions plancher et plafond.

3. $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$

La Figure 5.3 présente le graphique des fonctions plancher et plafond.

Ces deux fonctions sont accessibles dans `Python` en utilisant la librairie `math`, sous le nom de `floor` (*fonction plancher*) et `ceil` (*fonction plafond*).

```python
import math

print("Résultats de la fonction plafond")
print(math.ceil(1.4))
print(math.ceil(5.3))
print(math.ceil(-5.3))
print(math.ceil(22.6))
print(math.ceil(10.0))

print("Résultats de la fonction plancher")
print(math.floor(1.4))
print(math.floor(5.3))
print(math.floor(-5.3))
print(math.floor(22.6))
print(math.floor(10.0))
```

```
Résultats de la fonction plafond
2
6
-5
23
10
Résultats de la fonction plancher
1
5
```

```
-6
22
10
```

5.2 Fonctions en Python

DEVRAIT-ON PARLER DE ÇA????

DICTIONNAIRE, HACHAGE…

Exemple 5.3. Fonction de hachage dans Python

Hachage Python

Dictionnary in Python

A checksum is used to determine if something is the same.

If you have download a file, you can never be sure if it got corrupted on the way to your machine. You can use cksum to calculate a checksum (based on CRC-32) of the copy you now have and can then compare it to the checksum the file should have. This is how you check for file integrity.

A hash function is used to map data to other data of fixed size. A perfect hash function is injective, so there are no collisions. Every input has one fixed output.

A cryptographic hash function is used for verification. With a cryptographic hash function you should to not be able to compute the original input.

A very common use case is password hashing. This allows the verification of a password without having to save the password itself. A service provider only saves a hash of a password and is not able to compute the original password. If the database of password hashes gets compromised, an attacker should not be able to compute these passwords as well. This is not the case, because there are strong and weak algorithms for password hashing. You can find more on that on this very site.

TL;DR:

Checksums are used to compare two pieces of information to check if two parties have exactly the same thing.

Hashes are used (in cryptography) to verify something, but this time, deliberately only one party has access to the data that has to be verified, while the other party only has access to the hash.

5.3 Injection, surjection et bijection

Définition 5.3 (Fonction injective, surjective, bijective)**.** Soit $f : A \to B$ une fonction. On dit que

- f **est injective** si elle n'associe jamais la même image à deux éléments distincts:
$$\forall\ a_1 \in A,\ \forall\ a_2 \in A,\ (a_1 \neq a_2) \to (f(a_1) \neq f(a_2))$$
- f **est surjective** si son image est l'ensemble B au complet, c'est-à-dire si tous les éléments de B sont atteints:
$$\forall\ b \in B,\ \exists\ a \in A,\ f(a) = b$$
- f **est bijective** si elle est injective et surjective:
$$\forall\ b \in B,\ \exists! a \in A,\ f(a) = b$$

❗ Important

Si une fonction n'est pas **injective**, alors elle ne possède pas d'inverse.

❗ Important

Si une fonction n'est pas **surjective**, alors elle ne possède pas d'inverse.

Exemple 5.4. On considère un sous-ensemble f du produit cartésien de deux ensembles. Dans chaque cas, tracez son graphe saggital puis déterminez s'il s'agit d'une fonction ou non. De plus, si f est une fonction, déterminez si elle est injective, surjective ou bijective.

Ici, $L = \{a, b, c, d, e\}$, $M = \{a, b, c\}$, $C = \{1, 2, 3, 4\}$ et $D = \{1, 2, 3\}$.

a. $f = \{(1, a), (2, d), (3, c), (4, e)\} \subseteq C \times L$
b. $f = \{(1, a), (2, a), (3, c), (4, b)\} \subseteq C \times M$
c. $f = \{(1, a), (2, d), (3, c), (4, e), (1, b)\} \subseteq C \times L$
d. $f = \{(1, c), (2, a), (3, a), (4, a)\} \subseteq D \times M$
e. $f = \{(1, a), (2, a), (3, a), (4, a)\} \subseteq C \times L$

Exemple 5.5. La fonction $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ définie par $f(x_1, x_2) = x_1 + x^2$ est-elle oui on non injective? Est-elle oui ou non surjective? Est-elle oui ou non bijective?

5.3.1 Les dictionnaires dans `Python`

Le dictionnaire n'est pas une séquence mais un autre type composite. Ils ressemblent aux listes dans une certaine mesure (ils sont modifiables comme elles), mais les éléments que nous allons y enregistrer ne seront pas disposés dans un ordre immuable. En revanche, nous pourrons accéder à n'importe lequel d'entre eux à l'aide d'un index spécifique que l'on appellera une clé, laquelle pourra être alphabétique, numérique, ou même d'un type composite sous certaines conditions.

Exemple 5.6. Dites si le dictionnaire défini ci-dessous est une fonction injective, surjective, ou bijective.

```
jour = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"}
dejeuner = {"Oeufs", "Céréales", "Rôties", "Gruau", "Pâtisserie", "Jambon", "Crèpes"

mydict = {
    "Lundi": "Oeufs",
    "Mardi": "Céréales",
    "Mercredi": "Rôties",
    "Jeudi": "Gruau",
    "Vendredi": "Pâtisserie",
    "Samedi": "Jambon",
    "Dimanche": "Crèpes"
}
```

Exemple 5.7. Dites si le dictionnaire défini ci-dessous est une fonction injective, surjective, ou bijective.

```
jour = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"}
dejeuner = {"Oeufs", "Céréales", "Rôties", "Gruau", "Pâtisserie", "Jambon", "Crèpes"
```

```python
mydict = {
    "Lundi": "Oeufs",
    "Mardi": "Oeufs",
    "Mercredi": "Rôties",
    "Jeudi": "Gruau",
    "Vendredi": "Pâtisserie",
    "Samedi": "Jambon",
    "Dimanche": "Crèpes"
}
```

5.3.2 Fonction de hachage

Une **fonction de hachage** est une fonction qui associe des données de taille arbitraire à des valeurs de taille fixe. Les valeurs renvoyées par une fonction de hachage sont appelées valeurs de hachage, codes de hachage, résumés, signatures ou simplement hachages. Les valeurs sont généralement utilisées pour être les indices d'une table de taille raisonnable appelée table de hachage. Le hachage ou adressage de stockage dispersé est donc l'utilisation d'une fonction de hachage pour créer les indices d'une table de hachage.

Les fonctions de hachage sont utilisées dans les applications de stockage et de récupération de données pour accéder aux données en un temps réduit, en fait quasi-constant. Elles requièrent un espace de stockage à peine plus grand que l'espace total requis pour les données. Ainsi, le hachage est une forme d'accès aux données efficace en termes de calcul et d'espace de stockage.

L'intérêt des fonctions de hachage repose sur de bonnes propriétés statistiques. En effet, le comportement dans le pire des cas est mauvais, mais il se manifeste avec une probabilité extrêmement faible, en fait négligeable, et le comportement dans le cas moyen est optimal (collision minimale).

Une fonction de hachage est typiquement une fonction qui, pour un ensemble de très grande taille (théoriquement infini) et de nature très diversifiée, va renvoyer des résultats aux spécifications précises (en général des chaînes de caractère de taille limitée ou fixe) optimisées pour des applications particulières. Les chaînes permettent d'établir des relations (égalité, égalité probable, non-égalité, ordre…) entre les objets de départ sans accéder directement à ces derniers, en général soit pour des questions d'optimisation (la taille des objets de départ nuit aux performances), soit pour des questions de confidentialité.

Autrement dit : à 1 fichier (ou à 1 mot) va correspondre une signature unique (le résultat de la fonction de hachage).

> **!** Important
>
> Dans l'idéal, une fonction de hachage *devrait* être injective.

On peut trouver le haché d'un élément en `Python` en utilisant la commande `hash`. On peut remarquer dans le code ci-dessous que de changer une lettre minuscule en lettre majuscule (le F de fromage) change drastiquement le haché.

```python
phrase1 = "Maître Corbeau, sur un arbre perché, Tenait en son bec un fromage."
phrase2 = "Maître Corbeau, sur un arbre perché, Tenait en son bec un Fromage."

print(hex(hash(phrase1)), hex(hash(phrase2)))
```

```
-0x2514753287a6a7ee -0x69650207c616ddcb
```

6 Notation grand O

6.1 Mesurer un temps de calcul avec une fonction

6.2 Notation grand-O

6.3 Sommations

6.4 Établir la complexité d'un algorithme

6.5 Calculabilité et complexité

6.6 P vs NP

7 Introduction aux algorithmes

7.1 Bogo sort

```
from random import shuffle
from random import seed
from random import randint

def is_sorted(data) -> bool:
    """Determine whether the data is sorted."""
    return all(a <= b for a, b in zip(data, data[1:]))

def bogosort(data) -> list:
    """Shuffle data until sorted."""
    N = 0
    while not is_sorted(data):
        shuffle(data)
        N = N + 1
    return data, N

seed(1234)
N = 8
data = [randint(1,10) for x in range(N)]
bogosort(data)
```

```
([1, 1, 2, 2, 2, 2, 8, 10], 1552)
```

7.2 Exemples d'algorithmes

7.3 Fouille linéaire

7.4 Bubble sort

7.5 Insertion sort

7.6 Binary search

7.7 Heap sort

7.8 Complexité algorithmique

8 Théorie des nombres

8.1 Arithmétique modulaire

8.1.1 Division entière

8.1.2 Congruence modulo m

8.2 Entiers et algorithmes

8.2.1 Algorithme d'exponentiation modulaire efficace

8.2.2 Nombres premiers et PGCD

8.2.3 Algorithme d'Euclide et théorème de Bézout

8.2.4 Inverse modulo m

8.2.5 Résolution de congruence

8.2.6 Petit théorème de Fermat

8.3 Cryptographie à clé secrète

8.3.1 Chiffrement par décalage

8.3.2 Permutation de l'alphabet

8.3.3 Masque jetable

8.3.4 Chiffrement affine

8.4 Cryptographie à clé publique

8.4.1 Chiffrement RSA

9 Preuves et raisonnement mathématique

9.1 Méthodes de preuve

9.1.1 Preuve directe

Exemple 9.1. LE PRODUIT DE NOMBRES PAIRS ET IMPAIRS

Exemple 9.2. RACINE DE NOMBRES PAIRS

Exemple 9.3. PREUVE QUE n^2 EST PAIR

Exemple 9.4. Soit a, b et c des entiers. Si $a|b$ et $b|c$ alors $a|c$.

9.1.2 Preuve indirecte (par contraposée)

Exemple 9.5. Montrez que si n^2 est pair alors n est pair.

Exemple 9.6. Montrez que si $a + b$ est impair, alors a est impair ou b est impair.

Exemple 9.7. Soit p un nombre premier. Si $p \neq 2$ alors p est impair.

9.1.3 Preuve par contradiction

Exemple 9.8. EXISTE-T-IL UN PLUS PETIT NOMBRE RATIONNEL POSITIF?

Exemple 9.9. PREUVE QUE $\sqrt{2}$ EST IRRATIONNEL

Exemple 9.10. PREUVE QUE QU'IL EXISTE UNE INFINITÉ DE NOMBRES PREMIERS

Exemple 9.11. Il n'existe pas d'entiers x et y tels que $x^2 = 4y + 2$.

9.1.4 Principe des tiroirs de Dirichlet

Exemple 9.12 (Fonction de hachage)**.** Une fonction de hachage est une fonction qui transforme une suite de bits de longueur arbitraire en une chaîne de longueur fixe. Du fait qu'il y a plus de chaînes possibles en entrée qu'en sortie découle par le principe des tiroirs l'existence de collisions : plusieurs chaînes distinctes ont le même haché. Rendre ces collisions difficiles à déterminer efficacement est un enjeu important en cryptographie.

9.2 Principe de l'induction

9.2.1 Preuve par récurrence

Exemple 9.13. PREUVE QUE $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Exemple 9.14. PREUVE QUE $n < 2^n$

Exemple 9.15. PREUVE QUE 6 EST UN DIVISEUR DE $7^n - 1$

Exemple 9.16. MONTRER QUE NOUS POUVONS UTILISER DES T-GONES POUR REMPLIR UNE GRILLE $2^n \times 2^n$

Exemple 9.17. MONTRER QUE LA FACTORIELLE CROÎT PLUS RAPIDEMENT QUE L'EXPONENTIELLE

9.2.2 Algorithmes récursifs

9.2.2.1 Fonctions récursives

9.2.2.2 Algorithmes de type diviser pour régner

10 Dénombrement

10.1 Notions de base

10.2 Principe des nids de pigeon (principe des tiroirs de Dirichlet)

10.3 Permutations et combinaisons

10.4 Relations de récurrence et dénombrement

11 Graphes

11.1 Terminologie et types de graphes

11.2 Représentation des graphes

11.2.1 Représentation par listes d'adjacence

11.2.2 Représentation par matrice d'adjacence

11.3 Chemins dans un graphe

11.3.1 Chemins, circuits, cycles

11.3.2 Dénombrement de chemins

11.3.3 Chemins et circuits eulériens

11.3.4 Chemins et circuits hamiltoniens

11.4 Problème du plus court chemin

12 Arbres

12.1 Introduction aux arbres

12.2 Applications des arbres

12.3 Parcours d'un arbre

12.4 Arbres et tri

12.5 Arbres et recouvrement

12.6 Arbres générateurs de coût minimal

Références