Chapitre 9. Logique de base

9.1 Notions plus ou moins connues ...

9.1.1 Vocabulaire

9.1.2 Quantificateurs

Exercice 9.1.1
On considère 3 entiers différents v_1, v_2, v_3

1. Comment écrire à l'aide d'une expression booléenne

$$\forall i, v_i \text{ est pair}$$

2. Comment écrire à l'aide d'une expression booléenne

$$\exists i, v_i \text{ est impair}$$

3. Pourquoi ne fait-on pas comme cela, lorsque l'on a affaire à $\mathbb{N}$, $\mathbb{Z}$, ou $\mathbb{R}$ tout entier ?

Schéma de solution 1

1. Dire « tous les v_i sont pairs » c'est dire que « v_1 est pair et v_2 est pair et v_3 est pair » en `python` l'expression « x est pair » s'écrit `(x % 2 == 0)`, le « et » se note `and`. On écrira donc

```
(v1 %2 == 0) and (v2 % 2 == 0) and (v3 %2 == 0)
```

2. Dire « il y a un v_i imapir » c'est dire que « v_1 est impair ou v_2 est impair ou v_3 est impair » en `python` l'expression « x est pair » s'écrit `(x % 2 != 0)`, le « ou » se note `or`. On écrira donc

```
(v1 %2 != 0) or (v2 % 2 != 0) or (v3 %2 != 0)
```

3. Lorsque les valeurs à traiter sont nombreuses, voire quand il y en a un nombre infini, il est impossible de relier chaque propriété individuelle à l'aide d'un `and` ou d'un `or` Les quantificateurs universels et existentiels ne sont rien d'autre que des raccourcis d'écriture.

9.1.3 Négation

Exercice 9.1.2
On considère une liste d'entiers naturels et les deux propriétés suivantes

$$\forall x \in L, x \text{ est pair} \tag{9.1}$$

$$\exists x \in L, x \text{ est impair} \tag{9.2}$$

1. Parmi les 8 codes, le(s)quel(s) choisir pour résoudre la première propriété (Equation 9.1), et le(s)quel(s) choisir pour résoudre la seconde propriété (Equation 9.2) ?

```
def solveA(L:list) -> bool:
    i = 0
    while i < len(L) and L[i]%2 == 0:
        i = i + 1
    return (i == len(L))
```

```
def solveB(L:list) -> bool:
    i = 0
    while i < len(L) and L[i]%2 != 0:
        i = i + 1
    return (i == len(L))
```

```
def solveC(L:list) -> bool:
    i = 0
    while i < len(L) or L[i]%2 == 0:
        i = i + 1
    return (i == len(L))
```

```
def solveD(L:list) -> bool:
    i = 0
    while i < len(L) or L[i]%2 != 0:
        i = i + 1
    return (i == len(L))
```

```
def solveA2(L:list) -> bool:
    i = 0
    while i < len(L) and L[i]%2 == 0:
        i = i + 1
    return (i < len(L))
```

```
def solveB2(L:list) -> bool:
    i = 0
    while i < len(L) and L[i]%2 != 0:
        i = i + 1
    return (i < len(L))
```

```
def solveC2(L:list) -> bool:
    i = 0
    while i < len(L) or L[i]%2 == 0:
        i = i + 1
    return (i < len(L))
```

```
def solveD2(L:list) -> bool:
    i = 0
    while i < len(L) or L[i]%2 != 0:
        i = i + 1
    return (i < len(L))
```

2. Expliquez pourquoi, dans ces codes l'expression `(i < len(L))` est considérée comme la **négation** de `(i == len(L))` ?

3. Quelle autre expression aurait-on pu écrire en python ?

Schéma de solution 2
Lorsque vous devez analyser des codes, il est important de les tester sur quelques simples que vous maîtrisez. Ici on cherche à établir une propriété sur les éléments d'une liste, nous nous restreindrons à des listes d'entiers naturels, listes de petite taille (de 0 à 2 éléments)

1. `L = []`
2. `L = [1]`
3. `L = [2]`
4. `L = [1, 2]`
5. `L = [2, 1]`
6. `L = [2, 2]`
7. `L = [1, 1]`

Pour ces exemples nous souhaitons obtenir les réponses suivantes

exemples	Eq. 9.1	Eq. 9.2
`L = []`	**vrai**	**faux**
`L = [1]`	**faux**	**vrai**
`L = [2]`	**vrai**	**faux**
`L = [1, 2]`	**faux**	**vrai**
`L = [2, 1]`	**faux**	**vrai**
`L = [2, 2]`	**vrai**	**faux**
`L = [1, 1]`	**faux**	**vrai**

Attachons-nous maintenant à regarder la structure des codes proposés. Ils utilisent tous une boucle conditionnelle `while` qui autorise le passage à l'examen de l'élément suivant **si** le test est réussi. Une fois que le passage dans la boucle a été fait un certain nombre de fois, la fonction renvoie une expression booléenne qui compare la valeur de l'index i avec la longueur de la liste qui est examinée. Tous les codes proposés renvoient l'une des deux expressions

- `(i < len(L))` qui exprime que l'on a trouvé un élément de `L` à l'index i qui ne satisfait pas la condition de la boucle
- `(i == len(L))` qui exprime que l'on n'a pas trouvé d'élément de `L` qui ne satisfaisait pas la condition de la boucle.

Jetons maintenant un oeil sur la condition associée à la boucle. Dans les 8 codes proposés, on observe qu'il s'agit d'une expression booléenne combinant **2** expressions booléennes simples. La première est **toujours** `(i < len(L))` la seconde est un test de parité sur la valeur `L[i]`
Avant d'aller plus loin, il est important de s'attarder sur le choix de l'ordre de ces 2 expressions. Le test `(i < len(L))` est très important, il garantit que **si** l'évaluation de l'expression est **vraie**, alors il y a dans la liste `L` un élément à l'index `i`, et qu'écrire `L[i]` ne provoquera pas d'erreur à l'exécution – la relation est même plus forte, puisque la réciproque est vraie.
Le second aspect à examiner est la nature du lien entre les deux expressions booléennes du test, dans la moitié des cas on utilise `and`, le et, dans l'autre moitié on utilise le `or`, le ou.
Dans le chapitre « Premiers pas en python », nous avons étudié que

a	b	a and b
faux	**faux**	**faux**
faux	**vrai**	**faux**
vrai	**faux**	**faux**
vrai	**vrai**	**vrai**

a	b	a or b
faux	**faux**	**faux**
faux	**vrai**	**vrai**
vrai	**faux**	**vrai**
vrai	**vrai**	**vrai**

Ces tables mettent en lumière que lorsque `a` est **faux**, `a and b` sera toujours **faux**, il est donc inutile de chercher à calculer `b`. De même lorsque `a` est **vrai**, `a or b` sera toujours **vrai**, il est donc inutile de chercher à calculer `b`.
La conséquence de ce constat est que **tous** les codes proposés ici avec un `or` vont ignorer la seconde partie de l'expression (l'étude de la parité) lorsque `(i < len(L))`, mais plus grave encore lorsque cette expression est **fausse** vont chercher à tester la parité de `L[i]` et ainsi provoquer une erreur d'exécution avec le message d'erreur

```
IndexError: list index out of range
```

Il nous reste maintenant à regarder plus précisément les codes `solveA, solvB, solveA2, solveB2`

TO BE CONTINUED ...

9.1.4 Si ... Alors

Exercice 9.1.3
Pour chaque énoncé, donnez sa négation, sa réciproque et sa contraposée

1. Si T est un triangle rectangle, alors le carré de la longueur de l'hypothénuse est égal à la somme des carrés des longueurs des deux autres côtés.
2. S'il pleut, alors je prends un parapluie.

3. Si tout nombre pair supérieur à 3 se décompose comme une somme de 2 nombres premiers, alors tout nombre impair supérieur à 6 se décompose comme une somme de 3 nombres premiers.

4. Si tout nombre impair assez grand se décompose comme somme de 3 nombres premiers, alors tout nombre pair assez grand se décompose comme somme de 4 nombres premiers.

Schéma de solution 3

Rappel La négation de « si A alors B » est « A et non B » [a] ; la réciproque de « si A alors B » est « si B alors A » ; la contraposée de « si A alors B » est « si non B alors non A » les deux expressions signifiant la même chose.

a. En français, la préposition `mais` peut être utilisée à la place du `et` donnant « A mais non B ».

- « Si T est un triangle rectangle, alors le carré de la longueur de l'hypothénuse est égal à la somme des carrés des longueurs des deux autres côtés. »

  1. Le triangle T est rectangle et [mais] le carré de la longueur de l'hypothénuse n'est pas égal à la somme des carrés des longueurs des deux autres côtés
  2. Si le carré de la longueur de l'hypothénuse est égal à la somme des carrés des longueurs des deux autres côtés alors le triangle T est rectangle
  3. Si le carré de la longueur de l'hypothénuse n'est pas égal à la somme des carrés des deux autres côtés alors le triangle T n'est pas rectangle

- « S'il pleut, alors je prends un parapluie. »

  1. Il pleut et [mais] je ne prends pas un parapluie
  2. Si je prends un parapluie alors il pleut
  3. Si je ne prends pas de parapluie alors il ne pleut pas

- « Si tout nombre pair supérieur à 3 se décompose comme une somme de 2 nombres premiers, alors tout nombre impair supérieur à 6 se décompose comme une somme de 3 nombres premiers. »

  1. Tout nombre pair supérieur à 3 se décompose comme une somme de 2 nombres premiers et [mais] il existe un nombre impair supérieur à 6 qui ne se décompose pas comme une somme de 3 nombres premiers
  2. Si tout nombre impair supérieur à 6 se décompose comme une somme de 3 nombres premiers alors tout nombre pair supérieur à 3 se décompose comme une somme de 2 nombres premiers
  3. s'il existe un nombre impair supérieur à 6 qui ne se décompose pas comme une somme de 3 nombres premiers alors il existe un nombre pair supérieur à 3 qui ne se décompose pas comme une somme de 2 nombres premiers

- « Si tout nombre impair assez grand se décompose comme somme de 3 nombres premiers, alors tout nombre pair assez grand se décompose comme somme de 4 nombres premiers. »

  1. Tout nombre impair assez grand se décompose comme somme de 3 nombres premiers et [mais] il existe un nombre pair assez grand qui se décompose comme somme de 4 nombres premiers

2. Si tout nombre pair assez grand se décompose comme somme de 4 nombres premiers alors tout nombre impair assez grand se décompose comme la somme de 3 nombres premiers
3. S'il existe un nombre pair assez grand qui ne se décompose pas comme la somme de 4 nombres premiers, alors il existe un nombre impair assez grand qui ne se décompose pas comme la somme de 3 nombres premiers

La conjecture de Goldbach (1742) est « Tout nombre entier pair strictement supérieur à 2 peut être écrit comme la somme de deux nombres premiers. », vous pouvez consulter la page wikipedia sur cette conjecture et les quelques théorèmes en rapport – une conjecture est une propriété que l'on pense vraie mais dont on n'a pas la preuve.
« Si tout nombre pair supérieur à 3 se décompose comme une somme de 2 nombres premiers, alors tout nombre impair supérieur à 6 se décompose comme une somme de 3 nombres premiers. » est un énoncé vrai qui repose sur la propriété « si p est pair alors p+3 est impair ». Que le premier nombre pair supérieur à 3 est 4, que le premier nombre impair supérieur à 6 est 7, et que 3 est un nombre premier. •

Exercice 9.1.4 (Tâche de Wason)
On a disposé devant vous 4 cartes, sur la face exposée vous voyez respectivement

- un « A »,
- un « C »,
- un « 10 »,
- et un « 11 ».

L'expérimentateur vous explique qu'en fait un symbole est inscrit sur chacune des faces des cartes, et que la règle des écritures est « Si sur une face il y a une voyelle, alors sur l'autre face il y a un nombre pair ».
Votre tâche est de vérifier si la règle est bien valide pour toutes les cartes, en retournant certaines des cartes présentées.
Quelle(s) carte(s) retournez-vous ?

Schéma de solution 4
La règle « si sur une face il y a une voyelle, sur l'autre face il y a un nombre pair » peut être vérifiée quand on voit une voyelle, peut-être infirmer quand on voit un nombre impair. Il faut donc retourner la carte portant un « A » et la carte portant un « 11 ».

Exercice 9.1.5 (police)
Vous avez pour obligation de contrôler un débit de boissons et de dresser un procès-verbal en cas d'infraction à la loi qui stipule qu'« il est interdit de servir de l'alcool à un mineur ».
4 tables sont occupées par des personnes (vous masquant leurs consommations) et des verres (dont les consommateurs vont revenir) :

- à la première table une personne de 30 ans,
- à une autre une personne de 16 ans,
- à la troisième table un verre de boisson alcoolisée,
- et à la quatrième un verre de sirop à l'eau.

Qui contrôlez-vous ?

Les deux exercices (9.1.4, 9.1.5) sont identiques et pourtant les psychologues ont constaté que les réponses différaient. Voir l'article wikipédia sur la tâche de sélection de Wason.

Schéma de solution 5
C'est exactement le même exercice que la tâche de Wason, on vérifie ce que boit un(e) mineur(e), et on vérifie qui est la personne buvant de l'alcool.

9.2 Exercices de révision et compléments

Quantificateurs, Négation

Exercice 9.2.1 $(*)$
On se place dans $\mathbb{Z}$, pour chacun des énoncés, indiquez s'il est vrai, et donnez sa négation

1. $\forall a, \forall b, a+b=5$
2. $\exists a, \forall b, a+b=5$
3. $\forall a, \exists b, a+b=5$
4. $\exists a, \exists b, a+b=5$

Schéma de solution 6

Rappel La négation de $\forall a, p(a)$ est $\exists a, p(a)$ est faux. La négation de $\exists a, p(a)$ est $\forall a, p(a)$ est faux ou $\nexists a, p(a)$.

1. L'énoncé en français dit « quelque soit a, quelque soit b, la somme vaut 5 » ce qui est faux, par exemple $5+6 \neq 5$.
   on n'a pas $\forall a, \forall b, a+b=5$, est transformé en $\exists a$, non $\forall b, a+b=5$ qui est transformé en $\exists a, \exists b$, non $a+b=5$ qui donne finalement $\boldsymbol{\exists a, \exists b, a+b \neq 5}$

2. L'énoncé en français dit « on peut trouver un a, tel que quelque soit b, on ait $a+b=5$ » cet énoncé est faux, il n'existe pas de nombre dont la somme avec n'importe quel nombre soit constante. Dit autrement quand a est choisi, l'égalité $a+b=5$ fixe la valeur de b.
   On n'a pas $\exists a, \forall b, a+b=5$, est transformé en $\boldsymbol{\nexists a, \forall b, a+b=5}$ ou de manière équivalente en $\forall a$, non $\forall b, a+b=5$ qui est transformée en $\forall a, \exists b$, non $a+b=5$ qui donne finalement $\boldsymbol{\forall a, \exists b, a+b \neq 5}$.

3. L'énoncé en français dit « quelque soit a, il est possible de trouver b tel que $a+b=5$ » qui est vrai, il suffit en effet de prendre $b=5-a$ qui est toujours défini dans $\mathbb{Z}$.
   On n'a pas $\forall a, \exists b, a+b=5$ donne $\boldsymbol{\exists a, \nexists b, a+b=5}$, ou de manière équivalente $\boldsymbol{\exists a, \forall b, a+b \neq 5}$.

4. L'énoncé en français dit « il existe un a pour lequel on peut trouver un b tel que $a+b=5$ » qui est vrai. Cet énoncé est un cas particulier de l'énoncé précédent, en prenant $a=5$ on peut trouver b tel que $a+b=5$.
   On n'a pas $\exists a, \exists b, a+b=5$ donne $\boldsymbol{\nexists a, \exists b, a+b=5}$ ou de manière équivalente $\boldsymbol{\forall a, \nexists b, a+b=5}$ ou, de manière équivalente $\boldsymbol{\forall a, \forall b, a+b \neq 5}$.

Exercice 9.2.2 ($*$)
Pour chacun des énoncés suivants, indiquez s'il est possible d'établir sa véracité par une preuve directe (ou un contre-exemple), donnez l'énoncé contraire et indiquez si la réfutation est possible par une preuve directe (ou un contre-exemple).

1. $\forall x \in \mathbb{N}, x + 1 > x$
2. $\exists x \in \mathbb{N}, 2 \times x \leq x$
3. $\forall x \in \mathbb{N} - \{0\}, \forall y \in \mathbb{N} - \{0\}, x \times y \neq x + y$
4. $\exists x \in \mathbb{N}, \exists y \in \mathbb{N}, x \times y = x + y$
5. $\exists x \in \mathbb{N}, \forall y \in \mathbb{N}, x \times y \leq x + y$
6. $\forall x \in \mathbb{N} - \{0\}, \exists y \in \mathbb{N}, x \times y > x + y$

Schéma de solution 7

1. Dans $\mathbb{N}$, on peut enlever un nombre inférieur ou égal à lui-même. L'équation $x + 1 > x$ est équivalente à $1 > 0$ qui est vraie.
   L'énoncé contraire est $\exists x, x + 1 \leq x$. Pour les mêmes raisons que précédemment l'énoncé est équivalent à $1 \leq 0$ qui est faux.
2. En prenant $x = 0$ on obtient $0 \leq 0$ qui est vrai.
   L'énoncé contraire est $\nexists x \in \mathbb{N}, 2 \times x \leq x$ ou de manière équivalente $\forall x \in \mathbb{N}, 2 \times x > x$ une preuve directe de la fausseté de cet énoncé est $x = 0$ qui ne vérifie pas l'énoncé.
3. L'énoncé « Quelque soit un entier x non nul, quelque soit un entier y non nul, on a $xy \neq x + y$ » est faux. Puisque $x = 2 \neq 0$, $y = 2 \neq 0$ vérifie $2 \times 2 = 2 + 2$.
   L'énoncé contraire est $\exists x \in \mathbb{N} \setminus \{0\}, \exists y \in \mathbb{N} \setminus \{0\}, x \times y = x + y$
4. L'énoncé « on peut trouver un entier x, pour lequel on peut trouver un entier y, tel que $xy = x + y$ » est vrai, il suffit de prendre $x = y = 2$.
   L'énoncé contraire pourra avoir l'une des formes suivantes $\nexists x, \exists y, xy = x + y$, $\forall x, \nexists y, xy = x + y$ ou encore $\forall x, \forall y, xy \neq x + y$. Dont la fausseté est établie par le cas $x = y = 2$
5. L'énoncé « on peut trouver un entier x, tel que pour n'importe quel entier y, on ait $xy \leq x + y$ » est un énoncé vrai puisqu'on peut prendre $x = 0$ qui est élément absorbant pour la multiplication et élément neutre pour l'addition.
   L'énoncé contraire aura l'une des formes suivantes $\nexists x \in \mathbb{N}, \forall y \in \mathbb{N}, xy \leq x + y$ ou $\forall x \in \mathbb{N}, \exists y \in \mathbb{N}, xy > x + y$. Dont la fausseté est établie par $x = 0$
6. L'énoncé « quelque soit x un entier non nul, on peut trouver un entier y tel que $xy > x+y$ » est un énoncé faux, puisque quand $x = 1$, on cherche un entier y vérifiant $y > 1 + y$. On pourra s'amuser à prouver que l'énoncé est vrai pour $x \geq 2$.
   L'énoncé contraire prend l'une des formes suivantes $\exists x \in \mathbb{N} \setminus \{0\}, \nexists y \in \mathbb{N}, xy > x + y$, $\exists x \in \mathbb{N} \setminus \{0\}, \forall y \in \mathbb{N}, xy \leq x + y$.

Exercice 9.2.3 ($*$)
Y-a-t-il une différence entre les énoncés suivants ?

1. $\forall x \in \mathbb{R}^*, \exists y \in \mathbb{R}^*, x \times y = 1$
2. $\forall y \in \mathbb{R}^*, \exists x \in \mathbb{R}^*, x \times y = 1$

3. $\exists x \in \mathbb{R}^*, \forall y \in \mathbb{R}^*, x \times y = 1$

4. $\exists y \in \mathbb{R}^*, \forall x \in \mathbb{R}^*, x \times y = 1$

On s'intéressera plus précisément aux énoncés 1 et 2, 3 et 4, 1 et 3, 2 et 4. $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$.

Schéma de solution 8
• « pour tout réel non nul x, on peut trouver un réel non nul y, tel que le produit des deux vale 1 »
• « on peut trouver un réel non nul x tel qu'en le multipliant par n'importe quel réel non nul y, on trouve 1 »
Les deux premiers énoncés sont vrais, les deux derniers sont faux.
1 (resp. 2) On choisit x (resp. y) et on détermine y (resp. x) en divisant les deux membres de l'égalité par x (resp. y).
3 (resp. 4) On prend 2 cas particuliers pour y (resp. x), et on aboutit à $x = 0$ (resp. $y = 0$)

Exercice 9.2.4 ($*$)
Pour chacun des énoncés de l'exercice 9.2.3, écrire l'énoncé contraire (sa négation).

Schéma de solution 9
Ne sont détaillées que les étapes pour le cas 1 et le cas 3, les expressions attendues sont en gras

- $\exists x \in \mathbb{R}^*, \text{non } (\exists y \in \mathbb{R}^*, x \times y = 1)$ qui se réécrit soit en $\boldsymbol{\exists x \in \mathbb{R}^*, \nexists y \in \mathbb{R}^*, xy = 1}$ « On peut trouver un réel non nul x, tel qu'il n'y ait pas de réel non nul y pour lesquels leur produit vale 1 » soit en $\boldsymbol{\exists x \in \mathbb{R}^*, \forall y \in \mathbb{R}^*, xy \neq 1}$ « On peut trouver un réel non nul x, tel qu'en prenant n'importe quel réel non nul y, leur produit ne vale pas 1 »

- $\boldsymbol{\nexists x \in \mathbb{R}^*, \forall y \in \mathbb{R}^*, xy = 1}$ « On ne peut pas trouver de réel non nul x tel qu'en prenant n'importe quel réel non nul y, leur produit vale 1 » ou $\forall x \in \mathbb{R}^*, \text{non } (\forall y \in \mathbb{R}^*, xy = 1)$ qui se réécrit en $\boldsymbol{\forall x \in \mathbb{R}^*, \exists y \in \mathbb{R}^*, xy \neq 1}$. « Pour n'importe quel réel non nul x, on peut trouver un réel non nul y tel que leur produit ne vale pas 1 »