


Chapitre 3: Expressions booléennes



Une *variable booléenne* est une variable qui ne peut prendre que deux valeurs de vérité possibles: **True** ou **False**, matérialisées par les valeurs **1** et **0**, respectivement.

Si on considère des variables pour lesquelles la distribution de valeurs de vérité n'est pas binaire, on est dans le cas de la « logique floue ».

Exemple:

```
>>> var_bool = (2 < 5)
```

```
>>> print var_bool
```

True

```
>>> var_bool2 = (10%2 == 1)
```

```
>>> print var_bool2
```

False

```
>>>
```

UNIL | Université de Lausanne

02.10.12 Lausanne

UNIL - Informatique I

3.1 Opérateurs booléens unaires

Les *opérateurs booléens unaires* n'opèrent que sur une seule variable booléenne.

p une variable booléenne

0 => False et 1 => True

p	①p	②p	③p	④p
1	1	1	0	0
0	0	1	0	1

Opérateur **①**: identité

Opérateurs **②** et **③**: sans intérêt

Opérateur **④**: **négation** (non)
en logique: \sim ou \neg
en Python: **not**

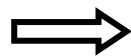
En logique, la négation se note « \sim » ou « \neg ».

On a donc:

Si P est fausse ($P=0$), alors $\neg P$ est vraie ($\neg P=1$).

Si P est vraie ($P=1$), alors $\neg P$ est fausse ($\neg P=0$).

Table de vérité



p	$\neg p$
1	0
0	1

En Python, la négation se note « **not** ».

Exemple:

```
>>> impair = True
>>> pair = not impair
>>> pair
False
>>>
```

3.2 Opérateurs booléens binaires

Les *opérateurs booléens binaires* opèrent sur deux variables booléennes.

Nous allons présenter les opérateurs suivants:

- conjonction (et)
- Disjonction (ou)

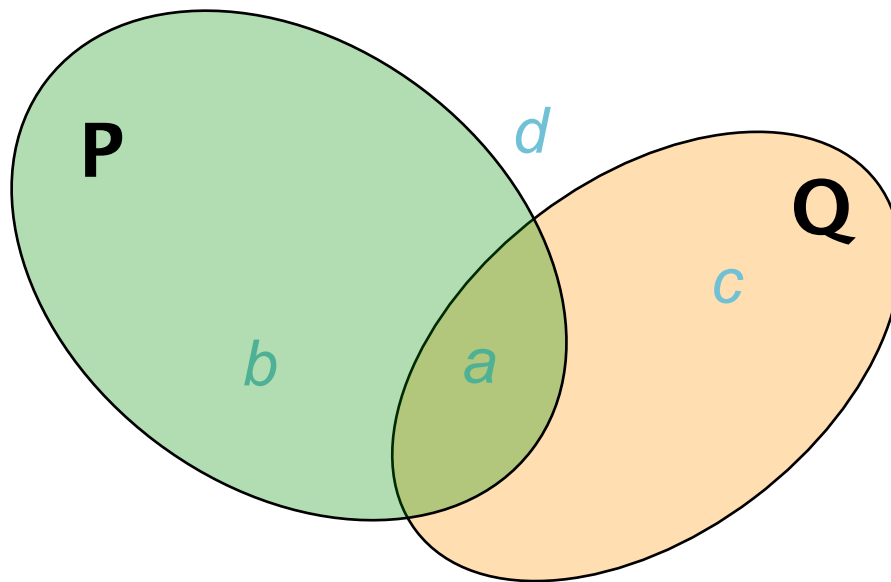
Conjonction: La conjonction représente le « et » logique.

En logique, on la note « \wedge »

Table de vérité \Rightarrow

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

En théorie naïve des ensemble, la conjonction représente l'intersection (notée \cap).



$$\{a\} \in P \cap Q$$
$$\{b, c, d\} \notin P \cap Q$$

En Python, on la conjonction s'écrit « **and** ».

Exemple:

```
>>> v1 = True
>>> v2 = True
>>> v1 and v2
True
>>> v2 = False
>>> v1 and v2
False
>>>
```

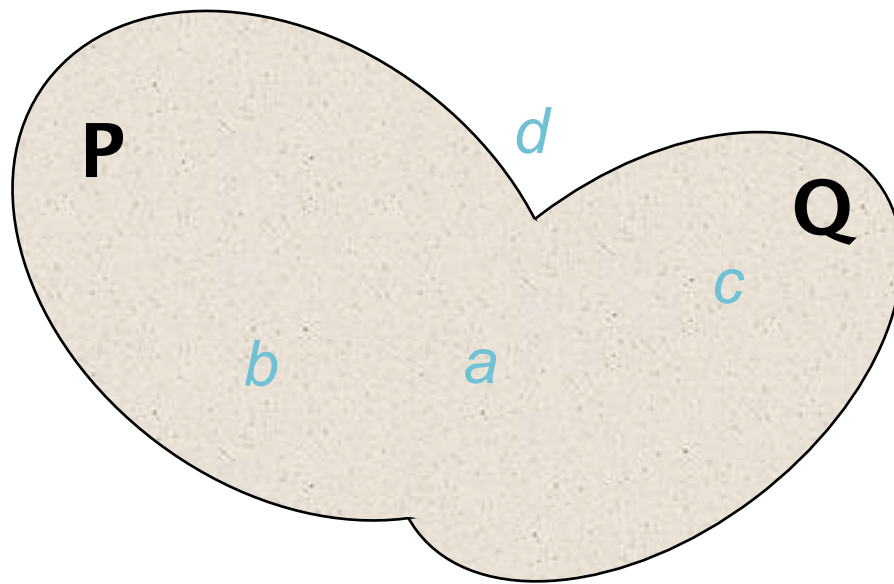
Disjonction: La conjonction représente le « ou » logique.

En logique, on la note « **v** »

Table de vérité \Rightarrow

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

En théorie naïve des ensemble, la conjonction représente la réunion (notée \cup).



$$\{a, b, c\} \in P \cup Q$$

$$\{d\} \notin P \cup Q$$

En Python, on la disjonction s'écrit « **or** ».

Exemple:

```
>>> v1=False
>>> v2=True
>>> v1 or v2
True
>>> v2=False
>>> v1 or v2
False
>>>
```

■ Lois de De Morgan

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

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

- Preuve de la première loi: $\neg (p \wedge q) \equiv \neg p \vee \neg q$

p	$p \wedge q$	q
1	1	1
1	0	0
0	0	1
0	0	0

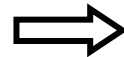


$p \wedge q$	$\neg(p \wedge q)$
1	0
0	1
0	1
0	1

identique
 \Rightarrow cqfd !

p	$\neg p$
1	0
1	0
0	1
0	1

q	$\neg q$
1	0
0	1
1	0
0	1



$\neg p$	$\neg p \vee \neg q$	$\neg q$
0	0	0
0	1	1
1	1	0
1	1	1

■ Evaluation conditionnelle de « and »

p	$p \wedge q$	q
1	1	1
1	0	0
0	0	1
0	0	0

⇒ donc inutile d'évaluer q.

⇒ q n'est évalué que si p est True et le résultat est la valeur de q.

si p est False ⇒ le résultat est False

■ Evaluation conditionnelle de « or »

p	$p \vee q$	q
1	1	1
1	1	0
0	1	1
0	0	0

si p est True \Rightarrow le résultat est True

\Rightarrow q n'est évalué que si p est False
et le résultat est la valeur de q.

3.3 Opérateurs de comparaison

Relations arithmétiques

Les *relations arithmétiques* sont des expressions qui utilisent des **<nombre>** comme opérandes et dont le résultat est un **<booléen>**.

En maths: = ≠ < ≤ > ≥

En Python: == != < <= > >=



en Python = est le symbole d'affectation!

Exemple:

```
>>> 2 < 3
```

```
True
```

```
>>> 2 <= 3
```

```
True
```

```
>>> 2 > 3
```

```
False
```

```
>>> 2 >= 3
```

```
False
```

```
>>> 2 != 3
```

```
True
```

```
>>> 2 == 3
```

```
False
```

```
>>>
```

```
>>> 2=3
```

```
File "<stdin>", line 1
```

```
SyntaxError: can't assign to literal
```

Remarque: il est possible d'enchaîner les comparaisons: $a < b < c$ est équivalent à: $a < b$ **and** $b < c$

Exemple 3-2: opérateurs de comparaison

$2 > 3 < 6 / 0$ est équivalent à

$2 > 3$ **and** $3 < 6 / 0$ et vaut False

il n'y a pas d'erreur en dépit du fait que l'on calcule $6 / 0$, puisque le membre de droite de l'opérateur **and** n'est pas évalué (voir paragraphe 3.2) .

Relations entre chaînes de caractères

Les *relations entre chaînes* sont des expressions qui utilisent des *<littéraux chaînes>* comme opérandes et dont le résultat est un *<booléen>*.

En Python: == != < <= > >=

- En Python, l'ordre sur de chaînes de caractères est l'ordre lexicographique des signes ASCII.
- Par exemple, "AB" < "AC", car "A"=="A" et "B" < "C"

36 \$	37 %	38 _	39 '	40 (41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;
60 <	61 =	62 >	63 ?	64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O	80 P	81 Q	82 R	83 S
84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [92 \	93]	94 ^	95 _
96 ,	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k



Exemple:

```
>>> date1 = "7 mars 2008"  
>>> date2 = '7 mars 2008'  
>>> date1 == date2  
True  
>>>
```

```
>>> date1 = "7 Mars 2008"  
>>> date2 = "7 mars 2008"  
>>> date1 == date2  
False  
>>> date1 > date2  
False  
>>>
```



3.4 Opérateur conditionnel

- L'opérateur « **if ... else ...** » est un opérateur ternaire qui correspond à une sorte d'instruction conditionnelle.
- La syntaxe est la suivante:
<expression1> if <condition> else <expression2>
- C'est une véritable instruction conditionnelle!
<expression1> et **<expression2>** sont des expressions et le type du résultat final peut être un **{<booléen>|<nombre>|<chaîne de caractères>}** qui dépend de l'évaluation des expressions.

Exemple 3-3: opérateur conditionnel

```
'Cout total: ' + str(n) + ' franc' + ('s' if n > 1 else '')
```

si n vaut 1 retourne la chaîne: Cout total: 1 franc

si n vaut 20 retourne la chaîne: Cout total: 20 francs