NSI - Première

Type simples : Booléens

qkzk

2022/01/01

Booléen

Booléen

En programmation, un booléen est un type de variable à deux états : *vrai* et *faux*.

Ils sont nommés ainsi d'après George Boole, fondateur de l'alèbre de Boole.

Booléen en Python

En Python, les booléens sont `True` et `False`, ils sont du type `bool`

```python
True
print(type(True))   # <class 'bool'>
False
print(type(False))  # <class 'False'>
```

Comparaison

Les opérateurs de comparaison courants sont identiques à ceux des mathématiques mais ATTENTION, il ne faut pas confondre l'égalité et l'affectation

```python
variable = 5    # une affectation
5 == 8          # une égalité (qui est fausse)
```

Le résultat d'une comparaison est toujours un booléen

Comparaisons des nombres

Comparaison	Symbole	Exemple	Résultat
Égalité	`==`	`1 + 2 == 3`	`True`
Différence	`!=`	`1 + 2 != 3`	`False`
Supérieur	`>`	`4 > 3`	`True`
Inférieur	`<`	`2.2 < 2 * 3`	`True`
Supérieur ou égal	`>=`	`5 >= 6`	`False`
Inférieur ou égal	`<=`	`8 <= 3`	`False`

Appartenance à une structure

On peut tester qu'un élément appartient à une structure avec le mot clé `in`

```python
"a"     in "bonjour"        # False
"bon"   in "bonjour"        # True
1        in [2, 3, 4]        # False
```

Opérations sur les booléens

Les opérateurs sur les booléens sont de deux types :

- opérateur unaire : prend *un* booléen et en renvoie *un*.
- opérateur binaire : prend *deux* booléens et en renvoie *un*.

Opérateur unaire : la négation

La négation: `not`

C'est le seul opérateur *unaire*, il donne le contraire de ce qu'on lui passe.

```python
not True    # s'évalue à False
not False   # s'évalue à True
```

Table de vérité avec `True` et `False`

`a`	`not a`
`True`	`False`
`False`	`True`

Table de vérité avec des bits

Les *tables de vérité* sont abbregées en notant :

- `1` pour `True`
- `0` pour `False`

`a`	`not a`
`1`	`0`
`0`	`1`

Opérateur binaire : le OU, noté `or`

Il est vrai si l'un des deux booléens est vrai.

```python
False or False  # False
False or True   # True
True or False   # True
True or True    # True
```

`a`	`b`	`a or b`
0	0	0
0	1	1
1	0	1
1	1	1

Opérateur binaire : le ET, noté `and`

Il est vrai si les deux booléens sont vrais.

```python
False and False  # False
False and True   # False
True and False   # False
True and True    # True
```

`a`	`b`	`a and b`
0	0	0
0	1	0
1	0	0
1	1	1

Opérateur binaire : le XOR noté ^

Il est vrai si EXACTEMENT un des deux booléens est vrai

```python
False ^ False  # False
False ^ True   # True
True ^ False   # True
True ^ True    # False
```

`a`	`b`	`a XOR b`
0	0	0
0	1	1
1	0	1
1	1	0

Python et les booléens

Python permet de comparer n'importe quoi à un booleen.

Par exemple, une chaîne de caractère vide est évaluée à fausse.

```python
bool(1)            # True
bool(0)            # False
bool("")           # False
bool("abc")        # True
bool([])           # False
bool([1, 2])       # True
```

- 0 est faux, les autres entiers sont vrais,
- une structure vide est fausse, les autres sont vraies.

Complément : `None` et l'identité `is`

Python propose la valeur `None` (rien) qui est fréquement utilisé pour représenter l'absence d'une valeur.

Étant le seul objet du type `NoneType`, on peut tester son *identité* avec `is` :

```
1 is None       # False
"abc" is None    # False
None is None    # True
a = 5
a is None       # False
```

On verra plus tard qu'une *fonction* qui ne se termine par `return ...` renvoie néanmoins `None`.

Table de vérité plus complexe

On peut chaîner les opérations booléennes pour construire une *expression booléenne.*

Une colonne par valeur et une colonne pour l'expression :

`a`	`b`	`c`	`(a and b) or (not c)`
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Propriétés mathématiques de l'algèbre de Boole

Il existe de nombreuses notations utilisées pour décrire l'algèbre de Boole, nous utiliserons les notations de Python.

Définition

On considère l'ensemble `{0, 1}` ou `{False, True}` muni de trois opérations :

la **négation** `not`, le **et logique** `and`, le **ou logique** `or`.

Elles sont définies par les tables de vérité présentées plus haut.

Complémentarité

- `not(not(a)) = a`
- `a or (not a) = 1`
- `a and (not a) = 0`

Associativité

- `a or (b or c) = (a or b) or c`
- `a and (b and c) = (a and b) and c`

Distributivité

- `a or (b and c) = (a and b) or (a and c)`
- `a and (b or c) = (a or b) and (a or c)`

Autres tables de vérité

Toutes les opérations binaires peuvent être définies à l'aide des trois opérateurs présentés plus haut.

Par exemple : `a xor b = (a and not b) or (not a and b)`