THÈME 3:
BOOLÉENS ET
CHAINES DE CARACTERES

Notions du thème:

- Chaîne de caractères : format et opérateurs
- Expressions booléennes

Précisions sur les chaînes de caractères

On a déjà utilisé les chaînes de caractères, notamment dans les fonctions `print()` et `input()`.

En Python, il existe 3 syntaxes pour les chaînes de caractères :

- avec des guillemets :

```
print("toto")
```

- avec des apostrophes :

```
Print('toto')
```

- avec des guillemets triples :

```
print("""toto""")
```

Intérêt de ces syntaxes

- On peut utiliser " dans une chaîne délimitée par ' ... '
- On peut utiliser ' dans une chaîne délimitée par "…"
- On peut utiliser " et ' dans une chaîne délimitée par """…"""
- """…""" permet aussi d'écrire des chaînes de caractères sur plusieurs lignes (on y reviendra plus tard)

Exemples

```
>>> print("C'est toto")
C'est toto
>>> print('C'est toto')
SyntaxError : invalid syntax
>>> print("Il a dit "hello" !")
SyntaxError : invalid syntax
>>> print('Il a dit "hello" !')
Il a dit "hello"
>>> print("""C'est toto qui a dit "hello" !""")
C'est toto qui a dit "hello" !
>>> print("""C'est toto qui a dit "hello""")
SyntaxError : …
```

Opérations sur les chaînes de caractères

- Longueur :

```
>>> s = "abcde"
>>> len(s)
5
```

- Concaténation :

```
>>> "abc" + "def"
'abcdef'
```

- Répétition :

```
>>> "ta " * 4
'ta ta ta ta'
```

Expressions booléennes
« ou » logique : `or`

- `expr1 or expr2` vaut vrai si et seulement si au moins une des deux expressions `expr1` et `expr2` est vraie.

- En Python, le « ou » est fainéant, c'est-à-dire que si la 1ère expression vaut **vrai**, la deuxième n'est pas évaluée

```
( 2 == 1 + 1 ) or (a > = 5)
```

ne provoque pas d'erreur même si `a` n'existe pas, le résultat vaut vrai

```
( 3 == 1 + 1 ) or (a > = 5)
```

provoque une erreur si `a` n'existe pas.

Expressions booléennes
« et » logique : `and`

- `expr1 and expr2` vaut vrai si et seulement si les deux expressions `expr1` et `expr2` sont vraies.

- En Python, le « et » est fainéant, c'est-à-dire que si la 1ère expression vaut **faux**, la deuxième n'est pas évaluée

```
( 2 > 8 ) and (a > = 5)
```

ne provoque pas d'erreur même si `a` n'existe pas, le résultat vaut faux

```
( 2 < 8 ) and (a > = 5)
```

provoque une erreur si `a` n'existe pas.

Lois de De Morgan

```
not(expr1 or expr2) = not(expr 1) and not(expr2)
```

Exemple :

```
not( a > 2 or b <= 4 )
```
équivaut à
```
( a <= 2 ) and ( b > 4 )
```

```
not(expr1 and expr2) = not(expr 1) or not(expr2)
```

Exemple :

```
not( a > 2 and b <= 4 )
```
équivaut à
```
( a <= 2 ) or ( b > 4 )
```

- Ces slides ont été réalisés par:
  - Amir Charif
  - Lydie Du Bousquet
  - Aurélie Lagoutte
  - Julie Peyre
- Leur contenu est placé sous les termes de la licence **Creative Commons CC BY-NC-SA**