

INFO-H-100 - Informatique

Séance d'exercices 3
Introduction à Python
Tests et fonctions

Université libre de Bruxelles
École polytechnique de Bruxelles

2013-2014

Composition de tests

Il y a trois **opérateurs logiques** : `and` (et), `or` (ou) et `not` (non).

On construit des expressions booléennes en utilisant les opérateurs de comparaison et logiques.

```
>>> x = 5
>>> 0 < x and x < 10
True
>>> x % 2 == 0 or x % 3 == 0
False
>>> not x > 10
True
```

En Python, il est possible de faire des comparaisons multiples :

```
>>> x = 5
>>> 0 < x < 10
True
```

Algèbre booléenne

- `a and b` est vrai si et seulement si `a` est vrai et `b` est vrai
- `a or b` est faux si et seulement si `a` est faux et `b` est faux

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Loi de De Morgan :

- `not (a and b)` est équivalent à `(not a) or (not b)`
- `not (a or b)` est équivalent à `(not a) and (not b)`

Exemple d'expressions équivalentes :

```
| not(0 >= x or x >= 1000)  
| 0 < x and x < 1000           #plus lisible
```

Test chaînés et imbriqués

On peut **chaîner** des conditions à l'aide de `elif` (else if) :

```
if x > y :  
    print('x est plus grand que y')  
elif x < y :  
    print('x est plus petit que y')  
else :  
    print('x est égal a y')
```

On peut également **imbriquer** des conditions à l'aide de l'indentation :

```
if x == y :  
    print('x est égal a y')  
else :  
    if x < y :  
        print('x est plus petit que y')  
    else :  
        print('x est plus grand que y')
```

Tuples

En Python, un **tuple** est une **séquence de valeurs** formée en séparant ces valeurs par des virgules. L'usage des parenthèses est recommandé.

```
>>> point = (1,2)
>>> print(point)
(1, 2)
```

```
>>> cours = ('INFO', 'H', 100)
>>> print(cours)
('INFO', 'H', 100)
```

On peut assigner les valeurs d'un tuple dans un autre possédant le même nombre de valeurs.

```
>>> (x,y) = point      #extraction
>>> print(x)
1
>>> print(y)
2
```

Tuples

Une fonction peut prendre un **tuple** en paramètre.

```
>>> def distance_origine(point):  
    (x,y) = point  
    return math.sqrt(x**2 + y**2)  
  
>>> distance_origine((0,1))  
1.0
```

Une fonction peut aussi **renvoyer un tuple**, ce qui lui permet par exemple de renvoyer plusieurs valeurs.

```
>>> def divise_modulo(a, b):  
    return (a/b, a%b)  
  
>>> (quotient, reste) = divise_modulo(5,2)  
>>> print(quotient, reste)  
2 1
```

Exercices

- 1 à 12.