Chapitre 1

Tests et conditions

« Ceci n'est pas un test ! »

1 Des outils pour comparer

Ce sont les *opérateurs de comparaison* :

Opérateur	Signification	Remarques
<	strictement inférieur	Ordre usuel sur `int` et `float`, lexicographique sur `str`…
<=	inférieur ou égal	Idem
>	strictement supérieur	Idem
>=	supérieur ou égal	Idem
==	égal	« avoir même valeur » *Attention :* deux signes =
!=	différent	
`is`	identique	être le même objet
`is not`	non identique	
`in`	appartient à	avec `str`, `list` et `dict`
`not in`	n'appartient pas à	avec `str` et `list` et `dict`

Python

```
>>> a = 2 # crée une variable de type int avec la
 ↪  valeur 2
>>> a == 2 # a vaut-elle 2 ?
```

```
True

>>> a == 3 # a vaut-elle 3 ?
False

>>> a == 2.0 # a vaut-elle 2.0 ?
True

>>> a is 2.0 # a est-elle la valeur 2.0 ?
False

>>> a != 100 # a est-elle différente de 100 ?
True

>>> a > 2 # a est-elle supérieure à 2 ?
False

>>> a >= 2 # a est-elle supérieure ou égale à 2 ?
True
```

Python

```
>>> a = 'Alice'
>>> b = 'Bob'
>>> a < b # a est il avant b dans l'ordre
 ↪  lexicographique ?
True

>>> 'ce' in a # 'ce' est-il une sous-chaîne de 'Alice'
 ↪  ?
True

>>> 'e' in b # 'e' est-il une sous-chaîne de 'Bob' ?
```

```
False

>>> liste = [1, 10, 100]
>>> 2 in liste # 2 est-il un élément de liste ?
False
```

Ces opérateurs permettent de réaliser des tests basiques. Pour des tests plus évolués on utilisera des « mots de liaison » logiques.

2 Les connecteurs logiques

- `and` permet de vérifier que 2 conditions sont *vérifiées simultanément*.
- `or` permet de vérifier qu'*au moins une* des deux conditions est vérifiée.
- `not` est un opérateur de *négation* très utile quand on veut par exemple vérifier qu'une condition est fausse.

Voici les tables de vérité des deux premiers connecteurs :

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

À ceci on peut ajouter que `not True` vaut `False` et vice-versa.

Python

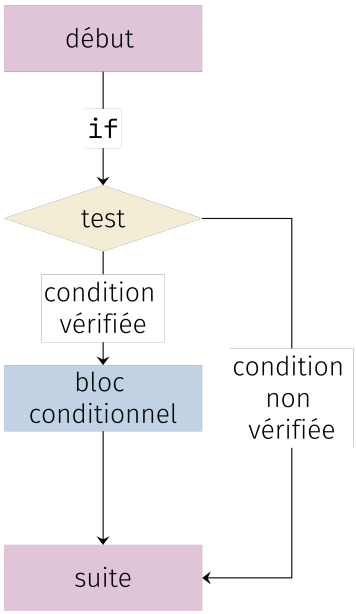
```
>>> True and False
False

>>> True or False
True

>>> not True
False
```

Python

```
>>> resultats = 12.8
>>> mention_bien = resultats >= 14 and resultats < 16
>>> print(mention_bien)
False
```

3 if, else et elif

Voici le schéma de fonctionnement d'un test `if` :



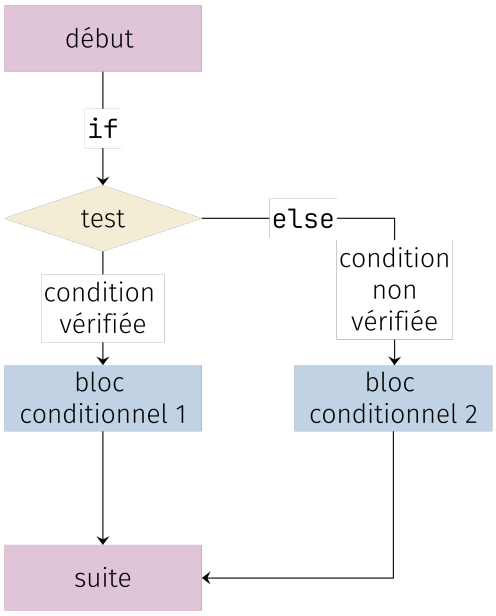
Attention : Un bloc conditionnel doit être *tabulé* par rapport à la ligne précédente : il n'y a ni `DébutSi` ni `FinSi` en PYTHON, ce sont les tabulations qui délimitent les blocs.

Python

```
phrase ='Je vous trouve très joli'
reponse = input('Etes vous une femme ?(O/N) : ')
if reponse == 'O':
```

```
    phrase += 'e' # remarquer la tabulation de cette
    ↪ ligne
phrase +='.'
print(phrase)
```

Voici le schéma de fonctionnement d'un test `if...else` :



Python

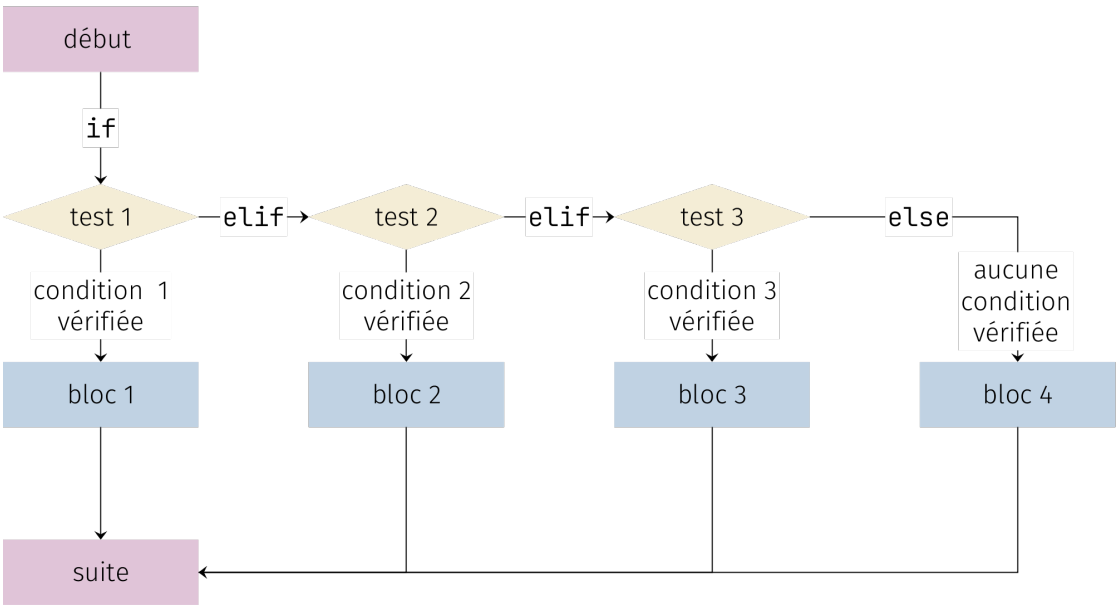
```
print('Bonjour')
age = int(input('Entrez votre age : '))
if age >= 18:
    print('Vous etes majeur')
else:
    print('Vous etes mineur.')
print('Au revoir.')
```

Voici un exemple de fonctionnement d'un test `if...elif...` :

Python

```
print('Bonjour')
prenom = input('Entrez un prénom : ')
if prenom == 'Robert':
    print("Robert, c'est le prénom de mon
    ↪ grand-père.")
elif prenom == 'Raoul':
    print("Mon oncle s'appelle Raoul.")
elif prenom == 'Médor':
    print("Médor, comme mon chien !")
else:
    print("Connais pas")
print('Au revoir.')
```

Et voici un schéma décrivant son fonctionnement :



On peut bien sûr inclure autant de `elif` que nécessaire.

4 Exercices

Exercice 1

Écrire un script qui demande son âge à l'utilisateur puis qui affiche `'Bravo pour votre longévité.'` si celui-ci est supérieur à 90.

Exercice 2

Écrire un script qui demande un nombre à l'utilisateur puis affiche si ce nombre est pair ou impair.

Exercice 3

Écrire un script qui demande l'âge d'un enfant à l'utilisateur puis qui l'informe ensuite de sa catégorie :

- trop petit avant 6 ans ;
- poussin de 6 à 7 ans inclus ;
- pupille de 8 à 9 ans inclus ;
- minime de 10 à 11 ans inclus ;
- cadet à 12 ans et plus ;

Exercice 4

Écrire un script qui demande une note sur 20 à l'utilisateur puis vérifie qu'elle est bien comprise entre 0 et 20. Si c'est le cas rien ne se produit mais sinon le programme devra afficher un message tel que `'Note non valide.'`.

Exercice 5

Écrire un script qui demande un nombre à l'utilisateur puis affiche s'il est divisible par 5, par 7 par aucun ou par les deux de ces deux nombres.

Exercice 6

En reprenant l'exercice du chapitre 1 sur les numéros de sécurité sociale, écrire un script qui demande à un utilisateur son numéro de sécurité sociale, puis qui vérifie si la clé est valide ou non.

Exercice 7

Écrire un script qui résout dans $\mathbf{R}$ l'équation du second degré $ax^2+bx+c=0$.
On commencera par `from math import sqrt` pour utiliser la fonction `sqrt`, qui calcule la racine carrée d'un `float`.
On rappelle que lorsqu'on considère une équation du type $ax^2+bx+c=0$

- si $a=0$ ce n'est pas une équation de seconde degré ;
- sinon on calcule $\Delta = b^2-4ac$ et
  - Si $\Delta<0$ l'équation n'a pas de solutions dans $\mathbf{R}$;
  - Si $\Delta=0$ l'équation admet pour unique solution $\frac{-b}{2a}$;
  - Si $\Delta>0$ l'équation admet 2 solutions : $\frac{-b-\sqrt{\Delta}}{2a}$ et $\frac{-b+\sqrt{\Delta}}{2a}$.

Pour vérifier que le script fonctionne bien on pourra tester les équations suivantes :

- $2x^2+x+7=0$ (pas de solution dans $\mathbf{R}$) ;
- $9x^2-6x+1=0$ (une seule solution qui est $\frac{1}{3}$) ;
- x^2-3x+2 (deux solutions qui sont 1 et 2).

Exercice 8

L'opérateur `nand` est défini de la manière suivante : si `A` et `B` sont deux booléens alors

`A nand B` vaut `not (A and B)`

Construire la table de vérité de `nand` en complétant :

A	B	A and B	not (A and B)
`False`	`False`		
`False`	`True`		
`True`	`False`		
`True`	`True`		