Leçon d'informatique : instructions conditionnelles

S. Benlhajlahsen - PCSI$_1$

Résumé On vous demandera fréquemment d'implémenter des algorithmes. Or, un algorithme, même simple, requiert couramment de devoir vérifier si une propriété est vérifiée ou non pour déterminer la marche à suivre. Vous disposez pour cela de mots clefs dont les noms sont parlants : `if` (si) et `else` (sinon). Il y en a un troisième, `elif` (contraction else if), qui permet d'écrire des "tests en cascade".

I Syntaxe

I.A Un seul test

Syntaxe La propriété que l'on veut vérifier s'appelle la *clause* ou la *condition*, et doit être à valeur booléenne. Si la clause vaut `True`, c'est le bloc indenté du if qui est exécuté, sinon c'est celui du `else` (voir figures 1 et 2).

```
if clause :
    instructions si clause vaut True
else :
    instructions si clause vaut False
```

FIGURE 1 – Syntaxe.

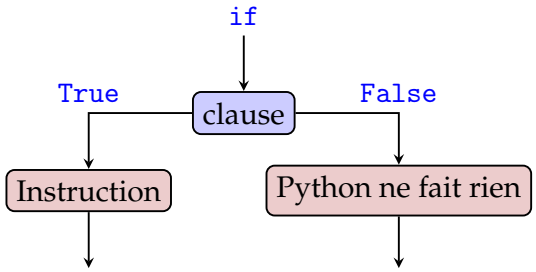

FIGURE 2 – Test conditionnel.

Remarque Le bloc `else` et son bloc indenté sont facultatifs. S'ils sont absents et que la clause vaut False, alors aucune instruction n'est exécutée et on passe à la suite.

Exemple : division euclidienne Reprenons la division euclidienne de a par b, mais en vérifiant qu'on ne divise pas par zéro (voir figure 3).

```
a = 5 # valeur exemple
b = 2 # valeur exemple
if b == 0 :
    print("Division par zéro !")
else :
    quotient = a//b
    reste = a%b
```

FIGURE 3 – Division euclidienne.

I.B Tests en cascade

Exemple : Si on cherche le signe d'un discriminant, on peut écrire :

- **si** $a > 0$, alors il y a deux racines réelles distinctes [a].
- **sinon, si** $\Delta < 0$, alors il y a deux racines complexes conjuguées [b].
- **Sinon** $\Delta = 0$ et il y a une seule racine réelle [c].

Cela se ramène à une successions (une "cascade") de tests booléens qui s'arrête au premier test donnant un résultat vrai.

a. (si = if)
b. (sinon si = else if = elif)
c. (sinon = else)

Syntaxe Lorsqu'une clause vaut `False`, `elif` (contraction de else if) permet d'enchaîner avec une autre clause. On peut avoir plusieurs blocs `elif` à la suite, et le bloc du `else` final joue le rôle d'instructions par défaut (exécuté si toutes les clauses valent `False`). On se reportera aux figures 4 et 5.

```
if clause1 :
    instructions si clause1 vaut True
elif clause2 :
    clause1 valant False , instructions si
        clause2 vaut True
else :
    clause1 valant False , instructions si
        clause2 valent False
```

FIGURE 4 – Syntaxe.

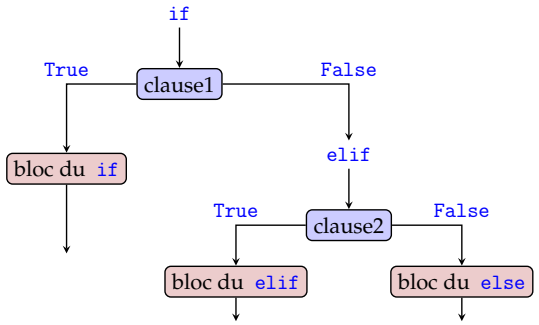

FIGURE 5 – Test conditionnel.

Exemple On veut déterminer la mention d'un éleve (voir figure 6).

```
def quelle_mention(note) :
    if note < 10 :
        return "Non admis"
    elif note < 12 :
        return "Admis sans mention"
    elif note < 14 :
        return "Mention assez bien"
    elif note < 16 :
        return "Mention bien"
    else :
        return "Mention très bien"

note = 15 # valeur exemple
mention = quelle_mention(note)
```

FIGURE 6 – Exemple.

Syntaxe Une fonction peut contenir plusieurs `return`, mais le premier rencontré met immédiatement fin à la fonction. Il est hautement recommandé que tous les return d'une même fonction renvoient le même type de variable, sans quoi le traitement du résultat sera compliqué !

Dans l'exemple de la figure 6 :

- Les différentes clauses sont testées les unes après les autres, dans l'ordre indiqué.
- La première clause égale à `True` met fin à toute la série de tests, de sorte qu'un seul `return` est exécuté.
- Le `else` sera atteint seulement si `note` est strictement supérieure à 16.

I.C Clauses combinées

Et logique : `and` On l'a déjà vu, deux clauses combinées avec `and` donnent un résultat égal à `True` si et seulement si les deux clauses valent `True`.
Par exemple, on veut savoir dans quel quadrant du plan (x, y) on se trouve le script de la figure 7.

Ou logique : `or` Un autre opérateur utile pour combiner deux clauses est le ou logique, `or` en Python : il donne `True` si au moins une des deux clauses vaut `True`. Autrement dit, il ne vaut `False` que si les deux clauses valent aussi `False`. Si on veut déterminer le nombre de jours dans un mois (en négligeant le problème des années bissextiles), on peut écrire la fonction de la figure 8.

```
1  def quel_quadrant(x,y) :
2      if x>0 and y>0 :
3          return  1
4      elif x<0 and y>0 :
5          return  2
6      elif x<0 and y<0 :
7          return  3
8      else :
9          return  4
10
11 x = -3.2 # valeur exemple
12 y = 9.7 # valeur exemple
13 quadrant = quadrant(x,y)
```

FIGURE 7 – Exemple.

```
1  def combien_jours(mois) :
2      if mois==2 :
3          return 27
4      elif mois==4 or mois==6 or mois==9 or mois==11 :
5          return 30
6      else :
7          return 31
8
9  mois = 4 # valeur exemple
10 nb_jours = combien_jours(mois)
```

FIGURE 8 – Exemple.

II Variable booléenne utilisée comme clause

Il n'y a rien de nouveau dans ce cas, mais l'utilisation de variables de type booléen peut conduire à du code surprenant quand on n'a pas l'habitude. Voyons d'abord le code trivial de la figure 9. Par construction, le bloc du `else` ne sera jamais exécuté [1]

Tester la valeur d'un booléen Soit `pluie`, variable booléenne qui vaut `True` s'il pleut, et `parapluie`, variable de type chaîne de caractères. On peut écrire l'exemple de la figure 10. Cependant, `pluie` est un booléen. On peut alors simplement écrire la fonction de la figure 11.

III Exercices

1. En fait, cet exemple n'est pas si ridicule que ça. Quand on écrit un programme assez long, il peut arriver qu'il contienne des parties "temporaires" que l'on veut pouvoir désactiver sans les effacer. On peut les englober dans un `if True` : tant qu'on veut les voir exécuter, puis le changer en `if False` : quand on ne le veut plus.

```
if True :
    print("Vérifié !")
else :
    print("Non vérifié !")
```

FIGURE 9 – Exemple.

```
def parapluie(pluie) :
    if pluie == True :
        return "oui"
    else :
        return "non"
pluie = True
precaution = parapluie(pluie)
```

FIGURE 10 – Exemple.

```
def parapluie(pluie) :
    if pluie :
        return "oui"
    else :
        return "non"
pluie = True
precaution = parapluie(pluie)
```

FIGURE 11 – Exemple.

III.A énoncés

Exercice 1 : Écrire une fonction `divisible2` qui prend en argument un entier `n` et qui permet de savoir si un nombre est divisible par 2.

Exercice 2 : Écrire une fonction `divisible2ou3` qui prend en argument un entier `n` et qui permet de savoir si un nombre est divisible par 2 ou 3.
Écrire une fonction `divisible2et3` qui prend en argument un entier `n` et qui permet de savoir si un nombre est divisible par 2 et 3.

Exercice 3 :

Soit $z = a + i \cdot b$ un nombre complexe d'argument θ[a]. On supposera que le script commence par `from math import atan, pi`[b]. Ecrire une fonction `arg(a,b)` qui renvoie l'argument. On souhaite déterminer son argument en suivant l'algorithme suivant :

- si $a > 0$ et
  - $b \neq 0$, alors $\theta = \arctan(b/a)$;
  - $b = 0$, alors $\theta = 0$.
- si $a = 0$ et
  - $b = 0$, alors $\theta = 0$;
  - $b > 0$, alors $\theta = \pi/2$.
  - $b < 0$, alors $\theta = -\pi/2$.
- si $a < 0$ et
  - $b = 0$, alors $\theta = \pi$;
  - $b > 0$, alors $\theta = \pi + \arctan(b/a)$;
  - $b < 0$, alors $\theta = -\pi + \arctan(b/a)$.

a. *modulo* 2π
b. La fonction `atan` correspond à la fonction arctangente.

Exercice 4 : Ecricre une fonction `est_lettre_minuscule` qui prend en argument un caractère `ch` et qui renvoie `True` ou `False` suivant que nature du caractère.
Indication : on utilisera les fontions de bases `chr` ou `ord`.
On écrire de la même manière une fonction `est_lettre_majuscule`.

Exercice 5 : Ecricre une fonction `change_casse` qui prend en argument un caractère `ch` et qui renvoie le caractère de la casse opposé. Ainsi, `change_casse('e')` doit renvoyer `'E'` et `change_casse('E')` doit renvoyer `'e'`. Si le caractère n'est pas alphabétique, celui doit être renvoyer à l'identique.

Exercice 6 : On considère un jeu de bataille navale simplifié : il ne s'agit pas de couler le porte-avions mais plutôt une

```python
def bataille(ligne,colonne):
    if ligne == ligne0 or colonne == colonne0:
        return "En vue"
    elif ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    else :
        return "À l'eau"
```

FIGURE 12 – Fonction bataille.

barque tenant sur une seule case repérée par ses coordonnées `ligne0` et `colonne0` qui sont des variables globales. On fait un tir sur une case repérée par ses coordonnées `ligne` et `colonne`. On veut alors le comportement suivant :

- si la barque est exactement sur la case considérée, le programme renvoie le texte "**Coulé**",
- si le tir atteint la bonne ligne ou la bonne colonne, le programme renvoie "**En vue**",
- si le tir est totalement raté, le programme renvoie "**À l'eau**".

On propose la fonction suivante de la figure 12.

1. Pourquoi cette fonction n'est-elle pas correcte (on pourra chercher le résultat si on a trouvé la bonne case).
2. Proposer une fonction valide.
3. Proposer une fonction qui n'utilise pas les opérateurs `and` et `or`.

Exercice 7 : Écrire une fonction qui étant donnée une équation du second degré $ax^2 + bx + c = 0$ identifiée par ses 3 coefficients réels, renvoie le nombre de solutions réelles. On supposera, sans avoir besoin de le vérifier, que a est non nul.

Exercice 8 :

On considère la fonction est définie par :

$$f : x \mapsto \begin{cases} -x+1 & \text{si} \quad x \leqslant 0 \\ 1 & \text{si} \quad x \in [0;2] \\ \frac{x}{2} & \text{si} \quad x \geqslant 2 \end{cases}$$

Dont on a donné une représentation graphique en figure 13 Écrire une fonction `fonction_morceau` qui calcule l'image d'un scalaire `x`.

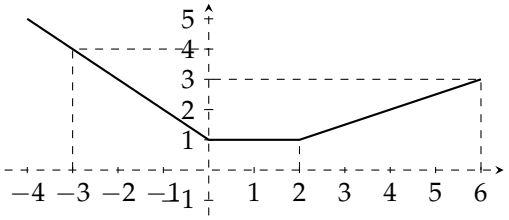

FIGURE 13 – Fonction définie par morceau.

III.B corrigés

1. voir figure 14 ;
2. voir figure 15 ;
3. voir figure 16 ;
4. voir figure 17 ;
5. voir figure 18 ;
6. (a) Si on calcule la fonction avec les bonnes coordonnées, la condition `ligne == ligne0 or colonne == colonne0` va être évaluée en `True` donc la fonction va retourner "En vue" au lieu de "Coulé".
   (b) Il suffit d'inverser les deux premières conditions (voir figure 19).
   (c) On peut imbriquer les conditions (voir figure 20). Remarquer le décalage vers la droite correspondant aux différentes indentations. L'imbrication des structures conditionnelles impose la mise en place de différents niveaux d'indentations qu'il faut respecter. Cette forme d'écriture utilisant des structures conditionnelles imbriquées est plus compliquée. L'usage des opérateurs booléens sera privilégié par la suite.
7. voir figure 21 ;
8. voir figure 22 ;

```python
def divisible2(n):
    if (n%2==0):
        return True
    return False
```

FIGURE 14 – Solution de l'exercice 1.

```python
def divisible2ou3(n):
    if (n%2==0) or (n%3==0):
        return True
    return False
def divisible2et3(n):
    if (n%2==0) and (n%3==0):
        return True
    return False
# Autre solution
def divisible2ou3(n):
    return (n%2 == 0) or (n%3 == 0)
def divisible2et3(n):
    return n%6 == 0
```

FIGURE 15 – Solution de l'exercice 2.

```python
def arg(a,b):
    if a>0:
        if b!=0:
            return(atan(b/a))
        else:
            return(0)
    elif a==0:
        if b==0:
            return(0)
        elif b>0:
            return pi/2
        else:
            return -pi/2
    else:
        if b==0:
            return(pi)
        elif b>0:
            return pi+atan(b/a)
        else:
            return -pi+atan(b/a)
```

FIGURE 16 – Solution de l'exercice 3.

```python
def est_lettre_minuscule(x):
    """ les lettres minuscules correspondent, dans la table ASCII, aux caracteres d'indice
        compris entre 97 et 122. """
    if ord (x) >=97 and ord(x) <=122:
        return True
    else :
        return False
def est_lettre_majuscule (x) :
    """ les lettres majuscules correspondent, dans la table ASCII, aux caracteres d'indice
        compris entre 65 et 90. """
    if ord (x) >=65 and ord(x) <=90:
        return True
    else :
        return False
```

FIGURE 17 – Solution de l'exercice 4.

```python
def change_casse(x):
    if est_lettre_minuscule(x):
        return chr(65+(ord(x)-97)%26)
    elif est_lettre_majuscule(x):
        return chr(97+(ord(x)-65)%26)
    else:
        return x
```

FIGURE 18 – Solution de l'exercice 5.

```python
def bataille(ligne,colonne):
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif ligne == ligne0 or colonne == colonne0:
        return "En vue"
    else :
        return "À l'eau"
```

FIGURE 19 – Fonction bataille.

```python
def bataille(ligne,colonne):
    if ligne == ligne0:
        if colonne == colonne0:
            return "Coulé"
        else:
            return "En vue"
    else:
        if colonne == colonne0:
            return "En vue"
        else:
            return "À l'eau"
```

FIGURE 20 – Fonction bataille.

```python
def nombreRacinesReelles(a, b, c):
    """Entrees : 3 flottants representant le polynome aX**2+bX+c, sortie : le nombre de
       racines reelles"""
    delta = b**2 - 4*a*c
    if delta > 0:
        return 2
    elif delta == 0:
        return 1
    else:
        return 0
# Autre version qui donne les racines sous forme liste
def racinesReelles(a, b, c):
    delta = b**2 - 4*a*c
    if delta > 0:
        return [(-b + delta)/(2*a), (-b - delta)/(2*a)]
    elif delta == 0:
        return [-b/(2*a)]
    else:
        return []
```

FIGURE 21 – Solution de l'exercice 7.

```python
def fonction_morceau(x):
    if x < 0:
        return -x + 1
    elif x < 2:
        return 1
    else:
        return x/2
# Autre solution
def fonction_morceau(x):
    return max(max(-x+1, 1), x/2)
```

FIGURE 22 – Solution de l'exercice 8.