

Troisième partie

Conditions

1 Introduction

Les langages de programmation permettent le calcul de valeurs booléennes.

Les valeurs booléennes sont indispensables pour avoir une exécution conditionnelle :

le choix de la prochaine instruction à exécuter ne se fera pas obligatoirement séquentiellement, mais pourra se faire en fonction de la validité d'une condition (i.e. $b > 5$).

Exemple (sur le portable : test.py)

Spécification 1.

$$\begin{aligned} f : \text{réel} &\rightarrow \text{réel} \\ x &\rightarrow 1/(x-2) \end{aligned}$$

Programmation 1.

```
def f (x):  
    return 1/(x-2)
```

Problème si $x = 2$: division par zéro. On va introduire une condition pour éviter ce type de problème. Pour cela nous allons avoir besoin d'expressions booléennes, de fonctions de comparaison et de fonctions booléennes.

2 Expressions booléennes

C'est une expression de type booléen (elle vaut vrai ou faux). Les plus simples sont **True** et **False**.

Les autres sont construites à l'aide de :

- opérateurs de comparaison ;
- opérateurs logiques ou booléens ;
- prédicats.

Les expressions booléennes permettent à leur tour de construire des *fonctions booléennes*, des *prédicats*, et des *expressions conditionnelles*.

Attention : en Python, tout élément à une valeur booléenne !

3 Opérateurs de comparaison

Un opérateur de comparaison est une expression booléenne à deux opérandes, qui permet de comparer des nombres (ou parfois, d'autres objets informatiques).

3.1 Égalité

En Python :

$$a == b \rightarrow \text{rend un booléen}$$

C'est un opérateur qui a deux opérandes de type quelconque et qui rend en résultat une valeur (vrai ou faux).

La sémantique de l'expression $e1 == e2$ est la suivante. Chacun des deux membres de l'expression est évalué, et si les résultats sont identiques, alors la valeur retournée est vrai (True). Si les deux résultats sont différents ou de types différents (sauf pour les nombres), la valeur est faux (False).

Exemple 3.1. Exemple sur le portable :

```
1 == 2  
1 == 1. 1==2/2  
1 == "a", 1=="1"
```


3.2 Différence

$a \neq b \rightarrow$ rend un booléen

est l'opposée de l'égalité.

Sur le portable, faire les différences sur les exemples précédents.

3.3 Opérateurs de comparaison

L'égalité et la différence peuvent prendre des arguments de type quelconque. Les opérateurs de comparaison également, mais à condition qu'une relation d'ordre soit définie.

Les deux opérandes peuvent être de types différents, mais le résultat est « rarement » intéressant, sauf pour les nombres !

$x < y \rightarrow$ rend un booléen

vaut vrai si `Eval(x)` est inférieur strictement à `Eval(y)`.

$x \leq y \rightarrow$ rend un booléen

vaut vrai si `Eval(x)` est inférieur ou égal à `Eval(y)`.

$x > y \rightarrow$ rend un booléen

vaut vrai si `Eval(x)` est supérieur à `Eval(y)`.

$x \geq y \rightarrow$ rend un booléen

vaut vrai si `Eval(x)` est supérieur ou égal à `Eval(y)`.

Exemple 3.2. Exemple sur le portable :

```
1 <= 2, 1 >= 1.  
True > False  
"accord" < "accueil" : c'est l'ordre lexicographique
```

4 Opérateurs booléens

On appelle opérateur booléen un opérateur dont les opérandes et le résultat sont de type `booléen`.

Les opérateurs booléens permettent d'effectuer des opérations sur des expressions booléennes, de la même façon qu'on peut effectuer des opérations entre des nombres entiers ou réels à l'aide de l'addition et de la multiplication.

Les opérateurs booléens les plus usuels sont la négation **non**, la conjonction **et** et la disjonction **ou**. Dans la suite de ce cours, nous ferons la distinction entre les fonctions booléennes « mathématiques » que nous noterons **non**, **et**, **ou** etc. et leur traduction en Python (**not**, **and**, **or**)???

4.1 La négation « non »

`non : booléen → booléen`

C'est un opérateur unaire! donne la négation d'une expression booléenne `x` : `non x` vaut vrai lorsque `x` est faux, et réciproquement.

En Python, cela est mis en œuvre par l'opérateur **unaire not**

Exemple 4.1. Exemple sur le portable :

```
not 1 == 2 (équivalent à not (1==2) et non à (not 1) == 2)  
not 3 >= 2**2 (équivalent à 3 < 2**2)
```


4.2 La conjonction « et »

$\text{et} : \text{booléen} \times \text{booléen} \rightarrow \text{booléen}$

On note x et y la conjonction de deux expressions. La valeur de x et y est **vrai**, si x vaut **vrai** et y vaut **vrai**. Sinon, x et y vaut **faux** (voir table 1).

x	y	x et y
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

TABLE 1 – Table de vérité de la conjonction

En Python, c'est l'opérateur **binaire** `and`

Exemple 4.2. Exemple sur le portable :

```
2<=3 and 3<=7 (ou 2<=3<=7 !!!)
not "aa" > "a" and 10%5!=0
not ("aa" > "a" and 10%5!=0)
```

4.3 La disjonction « ou »

$\text{ou} : \text{booléen} \times \text{booléen} \rightarrow \text{booléen}$

On note x ou y la disjonction de deux expressions. La valeur de x ou y est **vrai**, si x vaut **vrai** ou si y vaut **vrai** sinon elle est **faux** (voir la table 2).

x	y	x ou y
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

TABLE 2 – Table de vérité de la disjonction

En Python, c'est l'opérateur **binaire** `or`.

Attention à la priorité des ces trois opérateurs. Dans l'ordre croissant : **or**, **and**, **not**

Exemple 4.3. Exemple sur le portable : on cherche si une variable est divisible par 2 et soit par 3 ou par 5.

```
a%2==0 and a%3==0 or a%5==0 -> a=15 !
a%2=0 and (a%3==0 or a%5==0)
```

4.4 Prédicat de type

Un prédicat permet d'avoir des informations sur le type d'une expression.

Permet de tester si le premier paramètre est du type donné en second paramètre.

Valeur possible pour le type (pour le moment!) :

- `bool`;
- `int`;


```
— float;  
— str.
```

Exemple 4.4. Exemple sur le portable :

```
isinstance(2**64 - 2**64, int)  
isinstance(2**64 >= 0, bool)  
isinstance(2**64 and 0, bool)!!!!
```

5 Définition d'un prédicat

C'est une fonction dont le résultat est de type **booléen**. Il est d'usage de terminer son nom par un « Q », ou, en anglais, de commencer son nom par « is » ou « has ».

Exemple sur le portable.

Fonction **interieurQ** à trois paramètres réels (x , y et r) testant si le point (x,y) est à l'intérieur du cercle de centre $(0,0)$ et de rayon r . Sur le portable, fichier `interieurQ.py`

NB : r doit être de type **non négatif**. Après la spécification, donner au tableau la formule pour résoudre le problème

6 Expressions conditionnelles

La notion d'expression conditionnelle va nous ramener à notre exemple de départ. Une expression conditionnelle est une expression dont le résultat dépend de l'évaluation d'une ou plusieurs expressions booléennes. Les plus fréquemment utilisées sont l'expression **si** et l'expression **cas**.

6.1 L'expression « si »

Une expression **si** permet de choisir, en général, entre l'évaluation de deux expressions en fonction de la valeur d'une expression booléenne. En Python, cela est traduit par l'expression **if**.

Il en existe deux formes : avec ou sans « else », toutes deux étant des instructions composées.

Exemple 6.1. Cherchons – au tableau – à afficher si un nombre dépasse une limite (par exemple 100)

```
val = 99  
if (val >100):  
    print ("la valeur dépasse 100" )
```

Exemple 6.2. Cherchons – au tableau – à afficher si un nombre dépasse ou non une limite (par exemple 100)

```
val = 99  
if (val >100):  
    print ("la valeur dépasse 100")  
else:  
    print ("la valeur ne dépasse pas 100")
```

Exemple 6.3. Cherchons – au tableau – la fonction suivante (sans utiliser la fonction **max** de Python) :

$$\begin{array}{lll} \text{monMax} : \text{nombre} \times \text{nombre} & \rightarrow & \text{nombre} \\ x, y & \rightarrow & \text{le plus grand des nombres } x \text{ et } y \end{array}$$

En Python, cela s'écrit :

```
def monMax (a,b) :  
    "rend le plus grand de a et b"  
    if (a>b):  
        return a  
    else:  
        return b
```



```

monMax(2,3)
3
monMax(-2,-6)
-2

```

N.B. : en Python, cette fonction existe déjà, et s'appelle **max**. Son type est

$$\text{max} : \text{nombre} \times \dots \times \text{nombre} \rightarrow \text{nombre}.$$

6.2 L'expression « cas »

L'expression **cas** est une autre forme, très utilisée, d'expression conditionnelle. Son type est :

$$\text{cas} : (\text{booléen} \times \langle \text{type} \rangle) \times \dots \times (\text{booléen} \times \langle \text{type} \rangle) \rightarrow \langle \text{type} \rangle$$

En Python, elle est mise en œuvre grâce à l'instruction **if** utilisée avec le mot-clé **elif**.

La sémantique de cette instruction est la suivante. Si **Eval(cond1)** est **vrai**, le résultat est **expr1**. Si **Eval(cond1)** est **faux** et que **Eval(cond2)** est **vrai**, le résultat est l'évaluation de **expr2**, et ainsi de suite. Le résultat de l'expression est la valeur de la première expression **expr_i** dont la condition correspondante est **True**.

Exemple 6.4. Définissons une fonction **date** à un paramètre **an** de type entier, dont le résultat est une chaîne de caractères décrivant l'événement marquant associé à l'année **an** (voir le portable).

Spécification 2.

$$\begin{array}{ll}
 \text{date} : \text{entier} & \rightarrow \text{chaîne} \\
 \text{an} & \rightarrow \text{événement marquant date}
 \end{array}$$

Programmation 2.

```

def date (an): -> à donner au tableau
    if (an == 1515) :
        return "Bataille de Marignan"
    elif (an == 800) :
        return "Couronnement de Charlemagne"
    elif (an == 2000):
        return "fin du XXe siècle"
    elif (an > 2016):
        return "Je ne sais pas prédire l'avenir",
    else:
        return "Ca dépasse mes compétences..."

```

Test 1. `print(date(1515))`
 Bataille de Marignan
 `print(date(2.5))`
 Cela dépasse mes compétences!!!! Peut se corriger avec un `isinstance` au début!
 `print(date(2020)) => ajouter du test (cf date.py)`
 Je ne sais pas prédire l'avenir
 `print(date(1000))`
 Cela dépasse mes compétences...

Dans cet exemple, remarquer l'utilisation de la dernière « condition » (**else**) : elle permet de « récupérer » tous les arguments de type entier qui ne sont pas couverts par les conditions de l'expression **cas**.

Important : lorsque l'on imbrique des expressions composées, faire bien attention à l'indentation!
 Afficher sur le portable le fichier **testClassification**

7 Fonctions d'entrée / sortie

Quelques compléments sur la fonction `print` :

- si dans la liste des paramètres, vous ajoutez « `end=""` » l'affichage suivant se fera sur la même ligne ;
- le caractère « `\n` » provoque un retour à la ligne ;
- le caractère « `\` » annule l'effet du caractère suivant ;
- ...

La fonction `input()` provoque l'attente de la saisie d'une suite de caractères (terminée par « Entrée ») et rend comme résultat une **chaîne de caractères** correspondant à la saisie.

Si une chaîne de caractères est fournie en paramètre de `input()`, elle sera affichée avant la saisie.

Pour obtenir un nombre (entier ou réel) il faut utiliser des fonctions `int()` ou `float()` pour transformer la chaîne en nombre.

Faire exemple sur portable : fichier `testInput.py`