Algorithmique de base

Module : Fondement de la programmation

Douglas Teodoro

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

2019-2020

Sommaire

Objective

- Apprendre les instructions d'entrée et sortie
- Maîtriser le structure algorithmique de test
- Mettre en œuvre quelques structures pour résoudre des problèmes simples

Sommaire

SAISIE ET AFFICHAGE

Strings

- lettres, caractères spéciaux, espaces, chiffres
- mettre entre guillemets ou entre guillemets simples

```
hello = "bonjour"
```

- chaînes concaténées

```
nom = "ana"
message = hello + nom
message = hello + " " + nom
```

- effectuer certaines opérations sur une chaîne de caractères telle que définie dans la documentation Python https://docs.python.org

```
bete = hello + " " + nom * 3
```

Strings - Séquence d'échappement

séquence d'échappement	description	exemple
`\'`	apostrophe	`print("c\'est bon")`
`\"`	guillemet	`print("\"oui\": ...")`
`\\`	barre oblique inversée	`print("dir: C:\\Temp")`
`\a`	caractère d'appel audible	`print("alarm: \a")`
`\b`	retour arrière (backspace)	`print("hella\bo")`
`\n`	nouvelle ligne	`print("dir: C:\\Temp\name")`
`\r`	retour chariot	`print("ligne 1\rligne 2")`
`\t`	tabulation horizontale	`print("ligne 1\tligne 2")`

Affichage : `print`

- L'instruction **`print`** est utilisée pour afficher une information (texte, valeur, etc.) sur l'écran
- Elle prend à sa suite une chaîne de caractères, une valeur numérique, une variable, etc.
- Les informations à imprimer doivent être insérées entre parenthèses (syntax `fonction`)

```
print(...)
```

Affichage

Affichage

1. `a: int = 10`
2. `texte: str = "Hello World"`
3. `print(texte)`
4. `print("Bonjour IG")`
5. `print("nous sommes", a)`

```python
""" Affichage
données: ...
résultat: ...
"""

# déclaration et initialisation
# des variables
a:int = 10
texte: str = "Hello World"

# séquence d'opérations
print(texte)            # >> Hello World
print("Bonjour IG")     # >> Bonjour IG
print("nous sommes", a) # >> nous sommes 10
```

Saisie : input

- L'instruction **input** est utilisée pour inviter un utilisateur à rentrer au clavier une valeur
- Elle prend à sa suite une chaîne de caractères qui indique quel information sera saisie

```
input(...)
```

- Après exécuter cette instruction, l'programme attendra une entrée au clavier qui sera validée avec la touche d'entrée
- La valeur saisie peut affectée à une variable

```
var: str = input(...)
```

Saisie

Saisie

1. nom: str = input("Quel est votre nom ?")
2. print("Vous vous appelez", reponse)

```
""" Saisie
données: nom: str
résultat: affiche le nom
"""

# déclaration et initialisation
# des variables
reponse: str

# séquence d'opérations
reponse = input("Quel est votre nom ?") # >> Quel est votre nom ?
print("Vous vous appelez", reponse)     # >> Vous vous appelez <reponse>
```

Sommaire

LES TESTS ET CONDITIONS

Condition

Condition - Principe

Condition

Une **condition** est une affirmation : l'algorithme et le programme ensuite détermineront si celle-ci est vraie ou fausse

Exemple - Condition

- route : est en sens interdit
- devise : 1 CHF vaut moins que 1 EUR
- valeur : x > 10

En algorithmique, une expression évaluée est ou vraie (`True`) ou fausse (`False`)

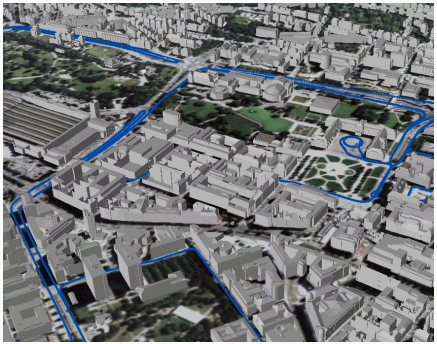
Definition

Principe de la condition : selon le résultat de l'évaluation d'une expression, l'algorithme va effectuer une action ou une autre

Condition - Principe

Exemple - Trajet

1. Allez tout droit
2. **Si** au prochain carrefour la route à droite est en sens interdit : continuez tout droit puis prenez à droite au carrefour suivant puis à gauche sur deux kilomètres jusqu'au rond-point
3. **Sinon** : tournez à droite et faites trois kilomètres jusqu'au rond-point
4. Au rond-point, **si** la sortie vers B est libre, prenez cette sortie
5. **Sinon**, prenez vers C puis trois cents mètres plus loin tournez à droite vers B



Comparaison

Une **condition** est souvent basée sur une comparaison

Comparaison

Une **comparaison** est une expression composée de trois éléments

- une première valeur : variable ou scalaire
- un opérateur de comparaison
- une seconde valeur : variable ou scalaire

```
valeur 1 OPERATEUR valeur 2
```

Les opérateurs de comparaison

Types de comparaison

égalité	x	`==`	y
différence	x	`!=`	y
inférieur	x	`<`	y
inférieur ou égal	x	`<=`	y
supérieur	x	`>`	y
supérieur ou égal	x	`>=`	y

Exemple - Comparaison

- **langage courant** : choisissez un nombre entre 1 et 10
- **mathématique** : $1 \leq nombre \leq 10$
- **algorithmique** :
  `nombre >= 1 and nombre <= 10`

Quasiment tout peut être testé en algorithmique (et en programmation) : les opérateurs s'appliquent sur quasiment tous les types de données, y compris les chaînes de caractères

Expression composée

Expression composée : plusieurs expressions peuvent **être liées entre elles** à l'aide d'un opérateur booléen, éventuellement regroupées avec des parenthèses pour pour modifier leur ordre de priorité

Exemple - Expression composée

```
(a == 1 or (b*3 == 6)) and c > 10
```

Quel est le résultat ?

- si `a = 1` et `c = 11`
- si `b = 2` et `c = 9.9`

Exercice - Condition

Question : Quelle est la valeur de la variable `condition` après l'exécution de l'algorithme `valeur_condition` ?

A) 5
B) True
C) -6
D) False

```
"""valeur_condition
données: les variables a, b et c
résultat: évaluation de l'expression
"""
# déclaration et initialisation
# des variables
a: int = 3
b: int = 1
c: int = -10
# séquence d'opérations
condition: bool = (a == 5 or (b*6 == 6)) and c > 10
print("condition:", condition)
```

Exercice - Condition

Question : Quelle est la valeur de la variable `condition` après l'exécution de l'algorithme `valeur_condition` ?

A) 5
B) True
C) -6
D) False

```
"""valeur_condition
données: les variables a, b et c
résultat: évaluation de l'expression
"""
# déclaration et initialisation
# des variables
a: int = 3
b: int = 1
c: int = -10
# séquence d'opérations
condition: bool = (a == 5 or (b*6 == 6)) and c > 10
print("condition:", condition)
```

PyCharm

exercice_1.py

Tests

Tests

En algorithmique, il n'y a qu'une seule instruction de test

`if` (ou si)

Le **test** permet d'exécuter un bloc d'instructions **si** la condition est vraie

- **Test simple** : instruction de contrôle du flux d'instructions qui permet d'exécuter une instruction sous condition préalable
- **Alternative simple** : instruction de contrôle du flux d'instructions qui permet de choisir entre deux instructions selon qu'une condition est vérifiée ou non
- **Alternative multiple** : instruction de contrôle du flux d'instructions qui permet de choisir entre plusieurs instructions en cascadant des alternatives simples

Tests - Structure algorithmique

Structure de contrôle effectuant un test et permettant un choix entre diverses parties du programme

Structure algorithmique

```
if condition : bloc
[elif condition : bloc]*
[else bloc]
```

- Les instructions entre crochets [...] sont facultatives
- Le * signifie que l'instruction peut être répétée 0 ou plusieurs fois
- On sort ainsi de l'exécution purement séquentielle des instructions

Structure du pseudo-code *vs.* Python pour le test

Algorithme : Test

Data : ...
Result : ...

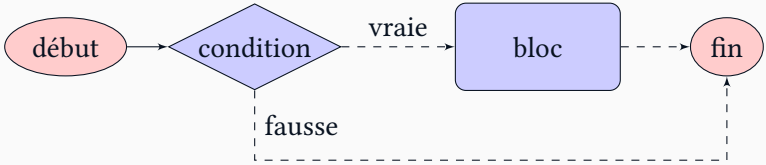
```
1  ...
   // séquence d'opérations
2  ...
3  if condition 1 then
4  |   instruction 1
5  |   ...
6  else if condition 2 then
7  |   instruction 2
8  |   ...
9  else
10 |   instruction 3
11 |   ...
```

```python
""" algo: test
données: ...
résultat: ...
"""
...
### séquence d'opérations
...
if condition_1:
    instruction_1
    ...
elif condition_2:
    instruction_2
    ...
else:
    instruction_3
    ...
```

Tests - Forme simple

Structure algorithmique

```
if condition : bloc
```



- La condition (ou expression) est du type booléenne
- La condition peut être représentée par une seule variable
  - si elle est égale à 0, elle représente le booléen `False`
  - sinon, le booléen `True`

Tests - Forme simple

Structure algorithmique

```
if condition : bloc
```

Que se passe-t-il si la **condition** est vraie ?

- Le bloc d'instructions situé après le **:** est exécuté
- La taille du bloc (le nombre d'instructions) n'a aucune importance : de une ligne à *n* lignes, sans limite
- Dans le cas contraire, le programme continue à l'instruction suivant

Exemple : valeur absolue d'un nombre

```
1  """ algo: abs
2  données: nombre
3  résultat: la valeur absolue
4  """
5  ### init. et decl. des
6  ### variables
7  nombre: int = -15
8
9  ### séquence d'opérations
10 if nombre < 0:
11     nombre = -nombre
12 print(nombre)
```

Tests - Forme simple

```
if condition : bloc
```

Expression simple

```
if x < 0 : y = 10

if x != y : y = x
```

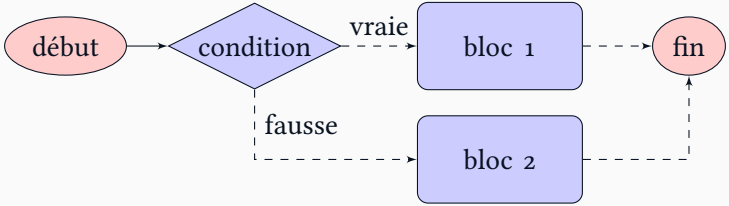
Expression composée

```
if (x > 0) and (x < 2) :
      y = 3*x

if (x <= 0) or (x >= 2) :
      y = 4*x
```

Tests - Alternative simple

Structure algorithmique

```
if condition : bloc 1
else: bloc 2
```



Utilisé dans le cas où il faut exécuter quelques instructions si la condition est fausse

Tests - Alternative simple

Structure algorithmique

```
if condition : bloc 1
else: bloc 2
```

Que se passe-t-il si la condition est vraie ?

- Le bloc d'instructions situé après le `:` (bloc 1) est exécuté
- Dans le cas contraire (condition est fausse), cette fois c'est le bloc d'instructions situé après le `else` (bloc 2) qui est exécuté

Alternative simple

```
if condition : bloc 1
else: bloc 2
```

Valeur absolue

```
if x < 0 :
      valeur_absolue = -x
else:
      valeur_absolue = x
```

Maximum

```
if x > y  :
      maximum = x
else:
      maximum = y
```

Tests - Alternative simple

Exemple : test de tri

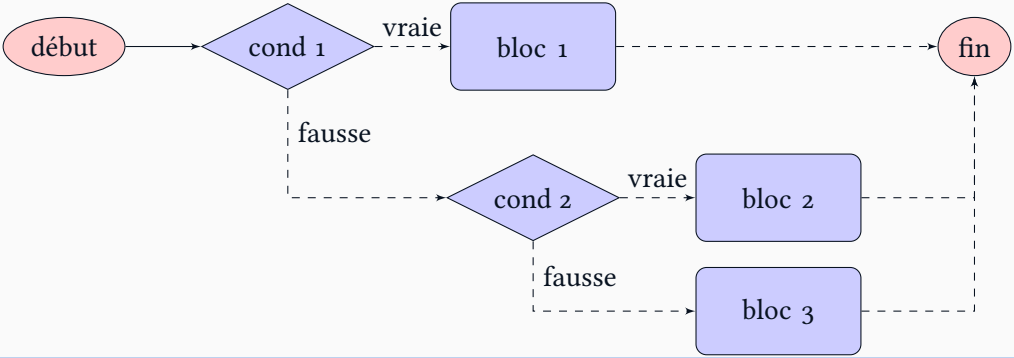
```
""" algo: test_tri
données: les variables x, y et z
résultat: affiche si les variables sont triées
"""
### déclaration de variables
x: float
y: float
z: float
### séquence d'opérations
x = float(input("Entrez la valeur de x"))
y = float(input("Entrez la valeur de y"))
z = float(input("Entrez la valeur de z"))

if z >= y and y >= x:
    print("Trié par ordre croissant")
else:
    print("Ces numéros ne sont pas triés")
```

Tests - Alternative multiple ou test imbriqué

Structure algorithmique

```
if condition 1 : bloc 1
elif condition 2 : bloc 2*
else: bloc 3
```

* signifie que l'instruction peut être répétée 0 ou plusieurs fois



Tests - Alternative multiple

Exemple : test de tri

```
""" algo: test_tri
données: les variables x, y et z
résultat: affiche si les variables sont triées
"""
### séquence d'opérations
x: float = float(input("Entrez la valeur de x"))
y: float = float(input("Entrez la valeur de y"))
z: float = float(input("Entrez la valeur de z"))

if z >= y and y >= x:
    print("Trié par ordre croissant")
elif z <= y and y <= x:
    print("Trié par ordre décroissant")
else:
    print("Ces numéros ne sont pas triés")
```

Exemple - Équation du deuxième degré

Résoudre une équation du deuxième degré :

$$ax^2 + bx + c = 0$$

Solution : $x = \frac{-b \pm \sqrt{\Delta}}{2a}$
où $\Delta = b^2 - 4ac$

- $\Delta > 0$: deux solutions
- $\Delta = 0$: une solution
- $\Delta < 0$: pas de solution

```python
import math
### déclaration et initialisation des variables
x1: float = None
x2: float = None
### séquence d'opérations
a: float = float(input("Entrez la valeur de a"))
b: float = float(input("Entrez la valeur de b"))
c: float = float(input("Entrez la valeur de c"))
delta: float = b**2 - 4*a*c
if delta > 0:
    x1 = (-b + math.sqrt(delta))/(2*a)
    x2 = (-b - math.sqrt(delta))/(2*a)
    print("Les solutions sont: x1=",x1,"et x2=",x2)
elif delta == 0:
    x1 = -b/(2*a)
    print("L'unique solution est :", x1)
else:
    print("L'équation n'a pas de solution")
```

PyCharm

exercice_2_[1,2].py

Conclusion

Contenu vu :

- instructions de saisie et affichage
- tests simple et imbriqué

Référence

Algorithmique - Techniques fondamentales de programmation
Chapitre : Tests et logique booléenne
Ebel et Rohaut, https://aai-logon.hes-so.ch/eni

Cyberlearn : 19_HES-SO-GE_631-1 FONDEMENT DE LA PROGRAMMATION (*welcome*)
http://cyberlearn.hes-so.ch

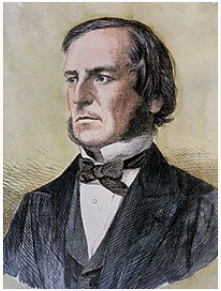
EXTRA

Sommaire

L’ALGÈBRE DE BOOLE

L'algèbre de Boole

- George Boole a développé l'algèbre qui porte son nom
- L'algèbre booléenne est utilisée en mathématique, logique, électronique et informatique
- Elle permet d'effectuer des opérations sur les variables logiques
- Comme son nom l'indique, elle permet d'utiliser des techniques algébriques pour traiter ces variables logiques



Prenez la phrase suivante absolument logique :

> une proposition peut être vraie OU fausse, mais ne peut pas être vraie ET fausse

Établir une communication

Communication téléphonique entre deux interlocuteurs

Une communication nécessite à la fois un émetteur et un récepteur

```
communication = émetteur ET récepteur
```

1. La communication est établie si l'émetteur appelle et que le récepteur décroche
2. Dans les autres cas, la communication ne s'établira pas, et sera donc FAUX

```
communication est FAUX si
```

- vous appelez quelqu'un (VRAI) mais qu'il ne décroche pas (FAUX)
- vous décrochez (VRAI) sans appel émis (FAUX)
- personne n'appelle ni ne décroche (FAUX dans les deux cas)

Table de vérité

```
communication = émetteur ET récepteur
```

émetteur	récepteur	communication
FALSE (n'appelle pas)	FALSE (n'appelle pas)	FALSE (pas de comm.)
FALSE (n'appelle pas)	TRUE (décroche)	FALSE (pas de comm.)
TRUE (décroche)	FALSE (n'appelle pas)	FALSE (pas de comm.)
TRUE (décroche)	TRUE (décroche)	TRUE (communication !)

émetteur	récepteur	communication
0	0	0
0	1	0
1	0	0
1	1	1

La loi AND (ET)

Énoncé

a ET *b* est VRAI si et seulement si *a* est VRAI **et** *b* est VRAI

La **loi AND** est aussi appelée la conjonction

Notation de la loi AND :

- $\cdot$ (le point) : $a \cdot b$
- $\wedge$: $a \wedge b$
- & , && ou *and* en programmation, selon les langages

AND		
a \ b	**0**	**1**
0	0	0
1	0	1

Associé à la multiplication arithmétique : $0.a = 0$

La loi OR (OU)

Énoncé

a `OU` *b* `est VRAI si et seulement si` *a* `est VRAI` **ou** *b* `est VRAI`

La **loi OR** est aussi appelée la disjonction

Notation de la loi OR :

- $+$ (signe plus) : $a + b$
- $\vee$: $a \vee b$
- `|` , `||` ou *or* selon les langages

OR		
a \ b	**0**	**1**
0	0	1
1	1	1

Associé à la addition arithmétique : $1 + a = 1$

La loi NOT (négation)

Énoncé

la négation de a est VRAI si et seulement si a est FAUX

Notation de la loi AND :

- $\bar{}$: $\bar{a}$
- $\neg$: $\neg a$
- ! ou *not* selon les langages

Propriétés : $a + \bar{a} = 1$ et $a \cdot \bar{a} = 0$

NOT		
a\ a	**0**	**1**
0	1	
1		0

Les propriétés

1. associativité : les termes peuvent être groupés de différentes façons
   - `a OR (b OR c) = (a OR b) OR c = a OR b OR c`
   - `a AND (b AND c) = (a AND b) AND c = a AND b AND c`
2. commutativité : l'ordre des variables logiques n'a aucune importance
   - `a OR b = b OR a`
   - `a AND b = b AND a`
3. distributivité :
   - `a AND (b OR c) = (a AND b) OR (a AND c)`
   - `a OR (b AND c) = (a OR b) AND (a OR c)`
4. complémentarité : la négation de la négation d'une variable logique est égale à la variable logique
   - `a = NOT (NOT a))`

Exercice - Boole I

Question : Quel est le 1^{er} affichage du programme `boole_1` ?

A) décroche

B) ne décroche pas

Algorithme : boole_1

Data : a,b,c,d : bool

```
1  a = TRUE
2  b = TRUE
3  c = FALSE
4  if ((NOT a) AND b) OR (a AND c) then
5  |   d = TRUE
6  else
7  |   d = FALSE
8  if d then
9  |   afficher "Je décroche"
10 else
11 |   afficher "Je ne décroche pas"
```

Exercice - Boole I

Question : Quel est le 1^{er} affichage du programme `boole_1` ?

A) décroche

B) ne décroche pas

Algorithme : boole_1

Data : a,b,c,d : bool

```
1  a = TRUE
2  b = TRUE
3  c = FALSE
4  if ((NOT a) AND b) OR (a AND c) then
5  |   d = TRUE
6  else
7  |   d = FALSE
8  if d then
9  |   afficher "Je décroche"
10 else
11 |   afficher "Je ne décroche pas"
```

Exercice - Boole II

Question : Quel est le 1^{er} affichage du programme `boole_2` si exécuté en python ?

A) décroche

B) ne décroche pas

C) pas d'affichage

Algorithme : boole_2

```
Data : a,b,c : bool
a = NONE
b = TRUE
c = TRUE
if ((NOT a) AND b) OR (a AND c) then
    afficher "Je décroche"
if NOT (((NOT a) AND b) OR (a AND c)) then
    afficher "Je ne décroche pas"
else
    afficher "Pas d'affichage"
```

Exercice - Boole II

Question : Quel est le 1^{er} affichage du programme `boole_2` si exécuté en python ?

A) décroche

B) ne décroche pas

C) pas d'affichage

Algorithme : boole_2

Data : a,b,c : bool

```
1 a = NONE
2 b = TRUE
3 c = TRUE
4 if ((NOT a) AND b) OR (a AND c) then
5     afficher "Je décroche"
6 if NOT (((NOT a) AND b) OR (a AND c)) then
7     afficher "Je ne décroche pas"
8 else
9     afficher "Pas d'affichage"
```