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

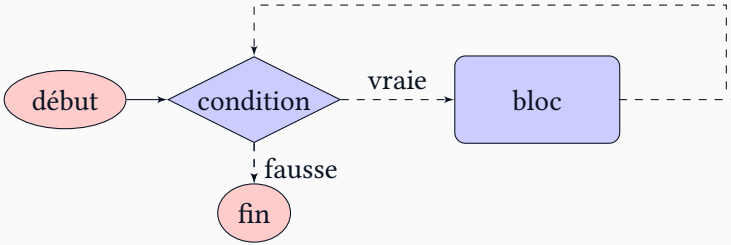
Objective

- Les boucles imbriquées
  - for
  - while
- Traitement des chaînes de caractères
- Mettre en œuvre quelques structures pour résoudre des problèmes simples

Sommaire

RAPPEL

Structure algorithmique - Tant que (While)



Acheter un téléphone

```
condition = avoir CHF 1500
total = CHF 0 (mois 0)
```

```
tant que total < 1500 faire
```

1. mois 1 : épargner CHF 300 (`total 300`)
2. mois 2 : épargner CHF 300 (`total 600`)
3. mois 3 : épargner CHF 300 (`total 900`)
4. mois 4 : épargner CHF 300 (`total 1200`)
5. mois 5 : épargner CHF 300 (`total 1500`)

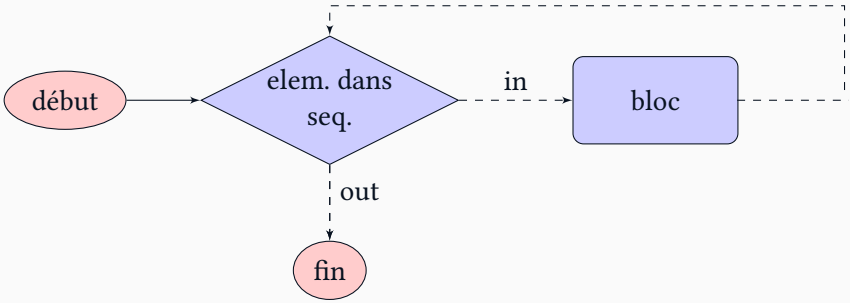
Tant que en Python : while

```
tant que condition faire
      bloc
```

Python - Tant que

```
while condition:
      # bloc d'instructions
      ...
```

Structure algorithmique - Pour (For)



Le structure `pour ... faire` : une boucle à l'usage des compteurs et séquences

Principe

À chaque passage dans la boucle, un **compteur est incrémenté ou décrémenté**, selon le cas :

→ on dit alors qu'il s'agit d'une structure incrémentale

Pour en Python - For

```
pour element ← debut à fin [pas] faire
    bloc
```

Python - Pour

```
for element in sequence:
    # bloc d'instructions
    ...
```

Exemples des séquences

- $1, 2, 3, ..., 9, 10$
  ```
  pour cpt ← 1 à 10 faire
  ```
- $0, 2, 4, ..., 8, 10$
  ```
  pour cpt ← 0 à 10 pas 2 faire
  ```
- $10, 9, 8, ..., 2, 1$
  ```
  pour cpt ← 10 à 1 pas -1 faire
  ```

Exemples des séquences

- $1, 2, 3, ..., 9, 10$
  ```
  for cpt in range(1,11):
  ```
- $0, 2, 4, ..., 8, 10$
  ```
  for cpt in range(0,11,2):
  ```
- $10, 9, 8, ..., 2, 1$
  ```
  for cpt in range(10,0,-1):
  ```

Équivalence Tant que - Pour

Algorithme : Tant Que

```
// séquence d'opérations
...
i = min
while i <= max do
    bloc
    i = i + pas
...
```

Algorithme : Pour

```
// séquence d'opérations
...
for i ← min to max pas pas do
    bloc
...
```

 # somme $1..n$

```
s: int = 0  # résultat
i: int = 1   # début
while i <= 10:
    s = s + i
    i = i + 1
```

 # somme $1..n$

```
s: int = 0 # résultat
for i in range(1,10+1,1):
    s = s + i
```

```
s: int = 0 # résultat
for i in [1,2,3,4,5,6,7,8,9,10]:
    s = s + i
```

Sommaire

LES BOUCLES IMBRIQUÉES

Des boucles imbriquées

Boucles imbriquées - une boucle dans une autre boucle

De même qu'une structure si ... alors peut contenir d'autres structures si ... alors, une boucle peut tout à fait contenir d'autres boucles

Calcule de la moyenne par étudiant-e

« prenons tou-te-s les étudiant-e-s du module 631-1 un-e par un-e »
« pour chaque étudiant-e, prenons toutes les notes et calculons la moyenne »

Voilà un exemple typique de boucles imbriquées :
→ on devra programmer une **boucle principale** et à l'intérieur, une **boucle secondaire**

Structures des boucles imbriquées

```
tant que condition_1 faire
    tant que condition_2 faire
        bloc
        ...
```

Python - Tant que imbriquée

```
while condition_1:
    while condition_2:
        # bloc d'instructions
        ...
```

```
pour i ← debut_1 à fin_1 faire
    pour j ← debut_2 à fin_2 faire
        bloc
        ...
```

Python - Pour imbriquée

```
for i in seq_1:
    for j in seq_2:
        # bloc d'instructions
        ...
```

Des boucles imbriquées - Exemple

Tables de multiplication 10 x 10

Calculer et afficher toutes les tables de multiplication de 1 à 10 :

- pour cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  seconde boucle : exécution de `1*1`, `1*2`, `1*3`, `...`, `1*9`, `1*10`

Des boucles imbriquées - Exemple

Tables de multiplication 10 x 10

Calculer et afficher toutes les tables de multiplication de 1 à 10 :

- pour cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1`, `1*2`, `1*3`, `...`, `1*9`, `1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1`, `2*2`, `2*3`, `...2*9`, `2*10`

Des boucles imbriquées - Exemple

Tables de multiplication 10 x 10

Calculer et afficher toutes les tables de multiplication de 1 à 10 :

- pour cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1, 1*2, 1*3, ..., 1*9, 1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1, 2*2, 2*3, ...2*9, 2*10`
- première boucle : table des **3**
  - seconde boucle : exécution de `3*1, 3*2, 3*3, ... ,3*9,3*10`

Des boucles imbriquées - Exemple

Tables de multiplication 10 x 10

Calculer et afficher toutes les tables de multiplication de 1 à 10 :

- pour cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1, 1*2, 1*3, ..., 1*9, 1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1, 2*2, 2*3, ...2*9, 2*10`
- première boucle : table des **3**
  - seconde boucle : exécution de `3*1, 3*2, 3*3, ... ,3*9,3*10`
- ...

Des boucles imbriquées - Exemple

Tables de multiplication 10 x 10

Calculer et afficher toutes les tables de multiplication de 1 à 10 :

- pour cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1, 1*2, 1*3, ..., 1*9, 1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1, 2*2, 2*3, ...2*9, 2*10`
- première boucle : table des **3**
  - seconde boucle : exécution de `3*1, 3*2, 3*3, ... ,3*9,3*10`
- ...
- première boucle : table des **10**
  - seconde boucle : exécution de `10*1, 10*2, 10*3, ..., 10*9, 10*10`

Toutes les tables de multiplication

```
""" algo: table_multiplication
données: nombre maximum pour calculer la multiplication
résultat: tables de multiplication
"""
### déclaration et initialisation
### des variables
TABLE_MAX: int = 10

### séquence d'opérations
for table1 in range(1, TABLE_MAX + 1):      # traite la boucle extérieure
    print("### table de", table1, "###")

    for table2 in range(1, TABLE_MAX + 1):  # traite la boucle intérieure
        resultat: int = table1 * table2
        print("%d x %d = %d" %(table1, table2, resultat))
```

Autre exemple - Carré

Créez un algorithme pour afficher un carré à l'écran comme ci-dessous :

```
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

```python
""" algo: carre
donées: n: int -> côté du carré
résultat: affiche un carré de taille n x n à l'écran
"""

### déclaration et initialisation des variables
n: int = None

### séquence d'opérations
n = int(input("Côté du carré ?"))

i: int = 1
while i <= n:            # traite les lignes
    j: int = 1
    while j <= n:        # traite les colonnes
        print("*", end="")
        j += 1
    print()
    i += 1
```

QCM 1

Question : Quel est le résultat de l'algorithme suivant :

A) 1234567

B) 123456

C) 23456

D) 234567

```
""" algo: boucle_x
données: jour -> int
affiche: ...
"""
# init. et decl. des variables
jour: int = 1

# séquence d'opérations
while jour < 7:
    jour += 1
    print(jour, end="")
```

QCM 1

Question : Quel est le résultat de l'algorithme suivant :

A) 1234567

B) 123456

C) 23456

D) 234567

```
""" algo: boucle_x
données: jour -> int
affiche: ...
"""
# init. et decl. des variables
jour: int = 1

# séquence d'opérations
while jour < 7:
    jour += 1
    print(jour, end="")
```

QCM 2

Question : Quel est le résultat de l'algorithme suivant :

A) 1

B) 9

C) 11

D) 13

```
""" algo: boucle_x
données: plus -> int
affiche: ...
"""
# init. et decl. des variables
plus: int = 1

# séquence d'opérations
while plus <= 10:
    plus += 2

print(plus)
```

QCM 2

Question : Quel est le résultat de l'algorithme suivant :

A) 1

B) 9

C) 11

D) 13

```
""" algo: boucle_x
données: plus -> int
affiche: ...
"""
# init. et decl. des variables
plus: int = 1

# séquence d'opérations
while plus <= 10:
    plus += 2

print(plus)
```

QCM 3

Question : Quel est le résultat de l'algorithme suivant :

A) 0-0-0-

B) 2-3-6-

C) 2-5-11-

D) 3-3-3-

```
""" algo: boucle_x
données: ...
affiche: ...
"""
# séquence d'opérations
for x in range(3, 0, -1):
    res: int = 0
    for y in range(3, 0, -1):
        res += y
    res = res/x
    print(str(int(res))+"-", end="")
```

QCM 3

Question : Quel est le résultat de l'algorithme suivant :

A) 0-0-0-

B) 2-3-6-

C) 2-5-11-

D) 3-3-3-

```
""" algo: boucle_x
données: ...
affiche: ...
"""
# séquence d'opérations
for x in range(3, 0, -1):
    res: int = 0
    for y in range(3, 0, -1):
        res += y
    res = res/x
    print(str(int(res))+"-", end="")
```

QCM 4

Question : Quel est le résultat de l'algorithme suivant :

A) 2-1-

B) 2-

C) 13-40-121-...

D) pas de affichage

```
""" algo: boucle_x
données: n -> int
affiche: ...
"""
# séquence d'opérations
n: int = 4
while n != 1:
    if n % 2 == 0:
        n = n//2
    else:
        n = 3*n + 1
    print(str(n)+"-", end="")
```

QCM 4

Question : Quel est le résultat de l'algorithme suivant :

A) 2-1-

B) 2-

C) 13-40-121-...

D) pas de affichage

```
""" algo: boucle_x
données: n -> int
affiche: ...
"""
# séquence d'opérations
n: int = 4
while n != 1:
    if n % 2 == 0:
        n = n//2
    else:
        n = 3*n + 1
    print(str(n)+"-", end="")
```

PyCharm

exercice_[1,2].py

Sommaire

CHAÎNES DE CARACTÈRES

Introduction

Nous allons maintenant nous intéresser au type string (ou str) qui permet de définir des chaînes de caractères

Exemple

```
cours:str = "Algo I"
```

Les chaînes de caractères - Fonctions utiles

len : calculer la longueur d'une chaîne
python : `len("Algo I")`

+ : concaténer 2 chaînes
python : `"Algo" + " I"`

lower : convertir la chaîne en minuscules
python : `"Algo I".lower()`

upper : convertir la chaîne en majuscules
python : `"Algo I".upper()`

split : diviser la chaîne
python : `p1, p2 = "Algo I".split()`

strip : supprimer les espaces excédantes au début et à la fin de la chaîne
python : `" Algo I ".strip()`

Traitements courants - Formatage

- **L'opérateur `%`** après une `string` est utilisé pour combiner une chaîne des caractères avec des variables

- L'opérateur **`%`** remplacera `%s` dans une chaîne de caractères par la variable `string` qui la suit

```
var = "def" ⟶ "abc %s" % var ⟶ "abc def"
```

- Le symbole spécial **`%d`** est utilisé comme caractère de remplacement pour les valeurs entiers

```
var = 10 ⟶ "abc %d" % var ⟶ "abc 10"
```

- Le symbole spécial **`%.␣f`** est utilisé comme caractère de remplacement pour les valeurs flottantes

```
var = 3.1415 ⟶ "abc %.2f" % var ⟶ "abc 3.14"
```

Traitements courants - Comparaison

Les opérateurs de comparaison s'appliquent aussi aux chaînes de caractères

Comparaison

```
012...789ABC...WYZabc...wyz

"A" < "Z" == True
"a" < "z" == True
"0" < "9" == True
"A" < "a" == True
"0" < "A" == True
"abc" < "b" == True
```

- Python a de nombreux types d'opérateurs de comparaison dont >=, <=, >, <=, >, <, etc.
- Les comparaisons donnent des valeurs booléennes : True ou False

Traitements courants - Indexation

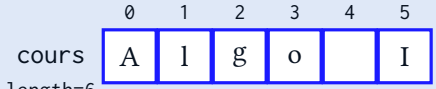
Définition

Une chaîne de caractères n'est rien d'autre qu'un tableau de caractères, où chaque caractère correspond à un indice

Exemple

Chaînes de caractères `cours` contenant 6 caractères :

```
cours: str = "Algo I"
```




```
cours[0] == 'A'
cours[1] == 'l'
cours[2] == 'g'
cours[3] == 'o'
cours[4] == ' '
cours[5] == 'I'
```

Traitements courants - Découpage

On peut utiliser l'**indice** pour accéder à une **sous-chaîne** d'une chaîne de caractères

`str[index]` : le caractère à l'indice `index`

`str[start:end]` : les caractères de l'indice `start` à l'indice `end-1`

`str[start:]` : les caractères de l'indice `start` jusqu'au reste du tableau

`str[:end]` : les caractères du début à l'indice `end-1`

Sous-chaîne

Pour la chaîne :

```
cours = "Algo I"
```

```
cours[0:2] == "Al"
cours[0:3] == "Alg"
cours[1:6] == "lgo I"
cours[4] == " "
```

Algorithme d'itération sur une chaîne de caractères

```
for i in range(len(<str>)):
    print(<str>[i])
```

```
for i in <str>:
    print(i)
```

```
var_str: str = "Algo I"
for i in range(len(var_str)):
    print("%d -> %s" % (i, var_str[i]))
```

```
0 -> A
1 -> l
2 -> g
3 -> o
4 ->
5 -> I
```

```
var_str: str = "Algo I"
for i in var_str:
    print("%s" % i)
```

```
A
l
g
o

I
```

PyCharm

exercice_[3-5].py

Conclusion

Contenu vu :

- les boucles imbrequées
- le traitement de chaînes des caractères

Référence

Algorithmique - Techniques fondamentales de programmation
Chapitre : Les boucles
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