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 le concept de variable algorithmique
- Comprendre les différents types
- Maîtriser les structures algorithmiques de séquence et test
- Mettre en œuvre quelques structures pour résoudre des problèmes simples

Sommaire

LES VARIABLES

Les variables

Dans un algorithme, on a toujours besoin de stocker provisoirement des valeurs

Definition

variable : objet informatique qui associe un nom à une valeur qui peut éventuellement varier au cours du temps

→ on affecte une nouvelle valeur au nom, d'où le nom de variable

Mémoire

devise	“CHF”
jour_mois	19
premier_cours	“àlgorithmique”
compteur	1

Composant d'une variable

Dans un algorithme, une **variable** possède :

- un nom
  ex. : `var_test`
- un type
  ex. : `var_test: int`
- une valeur
  ex. : `var_test: int = 10`

La **valeur** d'une variable :

- est fixée à un moment donné ex. : $t_1 \longrightarrow$ `var_test: int = 10`
- peut changer au cours du temps ex. : $t_2 \longrightarrow$ `var_test = 20`

Nom d'une variable

Definition

nom identificateur aussi explicite que possible

Comporte des lettres [`a...z`,`A...Z`,`_`] et des chiffres [`0...9`]

- exclut la plupart des signes de ponctuation, en particulier les espaces
- pas de lettres accentuées, de cédilles, de caractères spéciaux tels que $, #, @, etc.

Commence impérativement par une lettre ou `_`

La **casse** est importante : les caractères majuscules et minuscules sont distingués

- `python != pyThon != PYTHON`

Les types de variables

Les types de variables

Une variable peut stocker information des **différentes types** :

Quelques types primitifs

- **int** (entier) : `10`, `-2`, `80`
- **float** (réel) : `0.54`, `2.1`, `-1.5`
- **str** (chaîne des caractères) : `"algorithmique"`, `"programmation"`
- **bool** (booléen) : `True` (pour vrai) ou `False` (pour faux)
- **list** (tableau) : conteneur en 1 dimension ou plusieurs dimensions
  - `['a','b']`
  - `[['a','b'],['algo','prog']]`

Les nombres

Deux types **numériques** spécifiques sont utilisés en algorithmique pour representer les nombres :

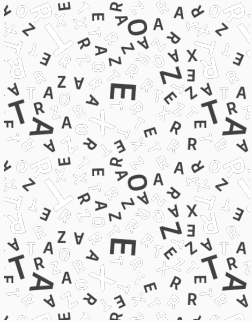
int	nombres sans virgule, négatifs ou positifs
float	nombres à virgule, positifs ou négatifs (utilise un point à la place de la virgule dans les langages)

Les caractères

Le type **`str`** (caractères) représente les caractère simples (ex. “a”, “$”) et les chaînes de caractères (ex. “CHF”, “ça va ?”)

Les caractères sont placées entre guillemets pour deux raisons :

- Éviter une ambiguïté entre les nombres sous forme de chaîne de caractères et les nombres au format numérique (ex., 1 contre “1”)
- Ne pas confondre le nom de la variable avec son contenu, notamment lors d’une affectation (ex., var_nom contre “var_nom”)



Caractères

```
nom_cours: str = "Algorithmique"
```

Le type booléen

Le type **bool** (booléen) est utilisé pour déterminer si une affirmation est vraie ou fausse

Il y a que deux valeurs possibles pour ce type : **True** ou **False**

Concept

L'affirmation $a > b$ selon laquelle variable a est supérieure à b

a) **si** a vaut 3 et b vaut 2, l'affirmation est vraie

b) **si** maintenant a vaut 1 et b vaut 2, l'affirmation est fausse

dans les 2 cas, $a > b$ est une expression qui **vaut soit vrai, soit faux**

Booléens

```
test: bool = True
est_active: bool = False
test_taille = a > b
```

QCM 1 - Nom de variables

Question : Dans la liste des variables ci-dessous, quels sont les noms acceptables ?

A) `Cours`
B) `algo`
C) `1erCours`
D) `_1_entree`
E) `2_entree`
F) `y`
G) `_1_entrée`
H) `algo prog`
I) `coursAlgo`
J) `étude`
K) `HEG`

QCM 1 - Nom de variables

Question : Dans la liste des variables ci-dessous, quels sont les noms acceptables ?

A) `Cours`
B) `algo`
C) `1erCours`
D) `_1_entree`
E) `2_entree`
F) `y`
G) `_1_entrée`
H) `algo prog`
I) `coursAlgo`
J) `étude`
K) `HEG`

QCM 2 - Types de variables

Question : Quel type de variable peut être utilisé pour représenter le nom d'une ville ?

A) int
B) str
C) bool
D) float

QCM 2 - Types de variables

Question : Quel type de variable peut être utilisé pour représenter le nom d'une ville ?

A) int
B) str
C) bool
D) float

QCM 3 - Types de variables

Question : Quel type de variable peut être utilisé pour représenter la distance entre Genève et Lausanne ?

A) float
B) bool
C) int
D) str

QCM 3 - Types de variables

Question : Quel type de variable peut être utilisé pour représenter la distance entre Genève et Lausanne ?

A) float
B) bool
C) int
D) str

Déclaration et affectation

La déclaration d'une variable

Format

```
nom_variable: type_variable
```

Avant d'être **utilisée**, une variable doit être déclarée et initialisée

```
""" Déclaration
données: ...
résultat: ...
"""
#### déclaration des variables
annee: int
nom_cours: str
nom_etudiant: str
note: float
est_premier_cours: bool
```

Affectation

Définition

Opération qui attribue une valeur à une variable

```
nom: type = valeur
   nom = valeur
```

Valeur d'une constante

`variable = constante` ex. : `solde: int = 10`

Valeur d'une expression

`variable = expression` ex. : `solde: int = x + 10`

Une fois **exécuté** l'instruction d'affectation, la variable est gardée dans la mémoire avec la valeur affectée

Affectation de valeurs

La valeur de la constante (droite) doit être du même type de la variable (gauche)

```
""" Affectation de valeurs
données: ...
résultat: ...
"""
### déclaration des variables
annee: int
nom_cours: str
nom_etudiant: str
note: float
est_premier_cours: bool

### initialisation des variables
annee = 2019
nom_cours = "Algorithmique de base"
nom_etudiant = "Douglas Teodoro"
note = 3.9
est_premier_cours = True
```

Affectation de valeurs

Dans la déclaration

Une valeur initiale ou par défaut peut être donnée à une variable lors de sa déclaration

Dans ce cas, l'opérateur d'affectation doit être utilisé lors de la déclaration

```
""" Déclaration et initialisation
données: ...
résultat: ...
"""
### déclaration et initialisation
### des variables
annee: int = 2019
nom_cours: str = "Algorithmique de base"
nom_etudiant: str = "Douglas Teodoro"
note: float = 3.9
est_premier_cours: bool = True
```

Affectation de variables

Affectation de variables

- Le principe est exactement le même pour l'affectation de valeurs
- La valeur de la variable à droite est affectée à la variable à la gauche

L'affectation de variables doit utiliser des variables de types compatibles

```
""" Affectation de variables
données: ...
résultat: ...
"""

### déclaration et initialisation
### des variables
annee: int = 2019

### séquence d'opérations
annee2: int = annee
```

Les constantes

constante cas particulier d'objet informatique, où le valeur ne change pas au cours du programme (c-à-d, invariable)

Déclaration d'une constante

- fait au début d'algorithme (*convention*)
- nom : utilise que de majuscule (*convention*)
- type et valeur : le même propriété de l'objet variable
- ex. : `COURS: str = "Algorithmique de base"`

« A constant is a type of variable whose value cannot be changed » (sic)

QCM 4 - Affectations

Question : Quelles seront les valeurs des variables a et b après l'exécution de l'algorithme `algo_x` ?

A) $a = 5$ et $b = 2$
B) $a = 2$ et $b = 5$
C) $a = 2$ et $b = 2$
D) $a = 5$ et $b = 5$

```
""" algo_x
données: les entiers a et b
résultat: ...
"""

### déclaration et initialisation
### des variables
a: int = 5
b: int = 2

### séquence d'opérations
a = b
b = a
```

QCM 4 - Affectations

Question : Quelles seront les valeurs des variables a et b après l'exécution de l'algorithme `algo_x` ?

A) $a = 5$ et $b = 2$
B) $a = 2$ et $b = 5$
C) $a = 2$ et $b = 2$
D) $a = 5$ et $b = 5$

```
""" algo_x
données: les entiers a et b
résultat: ...
"""

### déclaration et initialisation
### des variables
a: int = 5
b: int = 2

### séquence d'opérations
a = b
b = a
```

PyCharm

variables

Sommaire

INSTRUCTIONS

Instructions

Instruction

Commande élémentaire interprétée et exécutée par le processeur

Jeu d'instructions

Dans un processeur : **ensemble des instructions** que la puce peut exécuter

Bloc d'instructions

Dans un algorithme : **séquence d'instructions** pouvant être vue comme une seule instruction

```
""" algo_x
données: ...
résultat: ...
"""

### déclaration et initialisation
### des variables
a: int = 5
b: int = 2
resultat: float = 0.0

### séquence d'opérations
tmp: int = a*b + 1
resultat = tmp / 2
print(resultat)
```

Classes d'instructions algorithmique

- arithmétique : `+`, `-`, `*`, `/`, `%`
- logique : `not`, `and`, `or` (non, et, ou)
- contrôle du flux d'instructions : séquence, sélection (tests), itération (boucles), appels de procédure et fonctions
- entrée-sortie : `input`, `output`, `read`, `write` (saisir, afficher, lire, ecrire)
- transferts de données : `load`, `store`, `move`

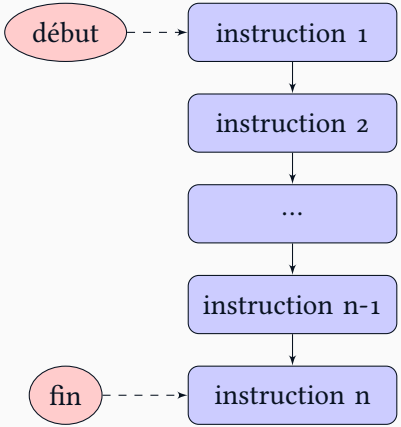
Séquence

Séquence

Definition

Suite d'instructions exécutées en séquence

1. `instruction 1`
2. `instruction 2`
3. `instruction 3`
4. `...`
5. `instruction n-2`
6. `instruction n-1`
7. `instruction n`



Opérateurs arithmétiques

Les opérateurs arithmétiques

Pour que les algorithmes puissent effectuer des calculs, il faut pouvoir faire des opérations simples. Vous utiliserez pour cela les symboles suivants :

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : modulo
- `//` : division entière
- `**` : puissance

Les opérateurs arithmétiques

Pour que les algorithmes puissent effectuer des calculs, il faut pouvoir faire des opérations simples. Vous utiliserez pour cela les symboles suivants :

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : modulo
- `//` : division entière
- `**` : puissance

Rappel : un modulo est le reste d'une division entière.
Par exemple, 15/2 vaut 7, mais il reste 1. On dit que 15 modulo 2 vaut 1.

Affectation d'expression

Valeur d'une expression

On évalue d'abord l'expression puis on affecte sa valeur à la variable

Suite arithmétique

1. `somme = 1 + 2 + ... + n-1 + n`
2. `somme = n*(n+1)/2`

```
""" évaluation d'une expression
données: n -> int
résultat: la somme des n 1ers entiers
"""
### déclaration et initialisation
### des variables
n:int = 5
somme: int = 0

### séquence d'opérations
somme = (1+n)*n/2
```

Affectation augmentée

Affectation augmentée

Le cas où la valeur attribuée dépend d'un précédent

Modification

- `i = i + 1` // incrémentation
- `i = i - 1` // décrémentation

```
""" affectation augmentée
données: les variables i, q et b
résultat: quelques affectations
"""
### déclaration et initialisation
### des variables
i: int = 0
q: float = 100
b: float = 2
### séquence d'opérations
i = i+1      # ou i += 1
i = i-1      # ou i -= 1
q = q/2      # ou q /= b
```

Affectation

- L'**affectation** est une opération typiquement informatique qui se distingue de l'égalité mathématique
- En **mathématique**, une expression du type `i = i+1` se réduit en `0 = 1` !
- En informatique,
  l'expression `i = i+1` conduit à ajouter `1` à la valeur de `i` (**évaluation de l'expression** `i+1`),
  puis à donner cette nouvelle valeur à `i` (**affectation**)

QCM 5 - Affectation

Question : Quelle est la valeur de la variable `i` après l'exécution de l'algorithme `i_mod` ?

A) 0
B) 1
C) 2
D) 3

```
""" i_mod
données: un indice i -> int
résultat: valeur d'i après les opérations
        d'affectation
"""

### déclaration et initialisation
### des variables
i: int = 0

### séquence d'opérations
i = i+1
i = i+1
i = i-1
i += 1
i -= 1
i -= 1
```

QCM 5 - Affectation

Question : Quelle est la valeur de la variable `i` après l'exécution de l'algorithme `i_mod` ?

A) 0
B) 1
C) 2
D) 3

```
""" i_mod
données: un indice i -> int
résultat: valeur d'i après les opérations
        d'affectation
"""

### déclaration et initialisation
### des variables
i: int = 0

### séquence d'opérations
i = i+1
i = i+1
i = i-1
i += 1
i -= 1
i -= 1
```

Opérateurs booléens

Les opérateurs booléens

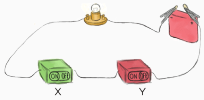
Opérateur **and**

Indique que **les deux expressions** situées avant et après l'opérateur doivent être toutes les deux vraies pour que l'ensemble le soit aussi :

```
var1 and var2
```

Soit `var1` et `var2` deux variables booléennes :

var1	var2	var1 and var2
False	False	False
False	True	False
True	False	False
True	True	True



Circuit en série : XY > X ET Y

X allumé, Y éteint > X ET Y est faux
> le courant ne passe pas

Les opérateurs booléens

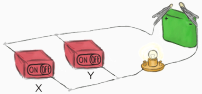
Opérateur **or**

Indique que **au moins une** des deux expressions, que ce soit celle située avant ou celle située après, doit être vraie pour que l'expression complète soit vraie aussi :

```
var1 or var2
```

Soit `var1` et `var2` deux variables booléennes :

var1	var2	var1 or var2
False	False	False
False	True	True
True	False	True
True	True	True



Circuit en dérivation : X+Y > X OU Y

X éteint, Y éteint > X OU Y est faux
> le courant ne passe pas

Les opérateurs booléens

Opérateur **not**

C'est **la négation** : si l'expression était vraie elle devient fausse, et vice versa :

```
not var
```

Soit `var` une variable booléenne :

var	not var
False	True
True	False

note	est_inf_seuil (var)	reussi (not var)
3.6	True	False
4.5	False	True

```
note: float
est_inf_seuil: bool = note < 4.00
reussi: bool = not est_inf_seuil
```

QCM 6 - Opérateurs logiques I

Question : Quel est le résultat de l'opération booléenne suivante pour

`a: bool = True` et `b: bool = False` ?

```
resultat = (not a) or b
```

A) `True`

B) `False`

QCM 6 - Opérateurs logiques I

Question : Quel est le résultat de l'opération booléenne suivante pour

`a: bool = True` et `b: bool = False` ?

```
resultat = (not a) or b
```

A) `True`

B) `False`

QCM 7 - Opérateurs logiques II

Question : Quel est le résultat de l'opération booléenne suivante pour

`a: bool = True` et `b: bool = False` ?

```
resultat = (a and (not b)) or b
```

A) `True`

B) `False`

QCM 7 - Opérateurs logiques II

Question : Quel est le résultat de l'opération booléenne suivante pour

`a: bool = True` et `b: bool = False` ?

```
resultat = (a and (not b)) or b
```

A) `True`

B) `False`

PyCharm

operations

Sommaire

CONCLUSION

Conclusion

Contenu vu :

- définition de variable (nom, type et valeur)
- déclaration et initialisation de variable
- instructions algorithmiques et évaluation d'expression
  - contrôle du flux d'instructions : séquence
  - arithmétique : `+`, `-`, `*`, `/`, `%`
  - logique : `and`, `or`, `not` (et, ou, non)

Référence

Algorithmique - Techniques fondamentales de programmation
Chapitre : Les variables et opérateurs
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