C.M. / T.D. Algorithmique avec Python

Sorbonne Université

GEAND – Tous les groupes

par Maxime Forriez

Télécharger l'AlgoBox

- https://www.xm1math.net/algobox/index.html
- https://www.xm1math.net/algobox/doc.html

Qu’est-ce qu’un algorithme ?

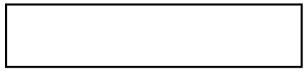
- Le mot « algorithme » vient du nom du mathématicien persan, Muhammad ibn Musa al-Khuwarizmi dit Al Khuwarizmi (IXe siècle), latinisé en Algoritmi.
- Un algorithme est une **recette**. Il s’agit d’une **suite d’instructions** qui, lorsqu’elles sont exécutées correctement, aboutissent au résultat attendu. C’est un énoncé en **langage clair**, bien défini et ordonné.
- Un algorithme décrit une **méthode de résolution** de problèmes courants, le plus souvent par le calcul.

Qu'est-ce qu'un algorithme ?

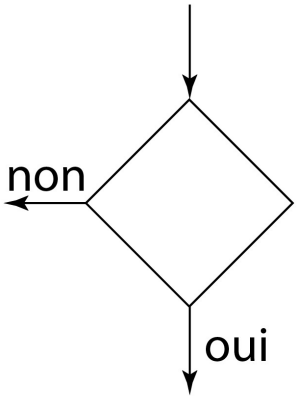
- Apprendre l'algorithmique (ou algorithmie), c'est apprendre à programmer dans les **règles de l'art**.
- L'algorithmique doit permettre le développement de **mécanisme de pensée par l'apprentissage** permettant d'optimiser les traitements que le développeur doit programmer.
  - Vitesse de calcul
  - Occupation de la mémoire
  - Quantité de lignes de programmation
  - *etc*.

Organigramme

début ou fin ou interruption



tâche à réaliser, instruction, opération

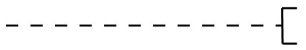


test un embranchement

entrée ou sortie

symbole de jonction, renvoi

liaison entre les éléments



commentaire

Le langage de description d'algorithmes (L.D.A.)

- Un **pseudo-code** fait référence à une **syntaxe ressemblant à un code**.
  - Il indique le fonctionnement d'une syntaxe de code.
  - Il illustre la conception d'un élément d'architecture de code.
  - Il ne fait aucune référence à un quelconque langage de programmation.
- Le pseudo-code permet de décrire avec **plus ou moins** de détails l'algorithme.
  - Il permet de passer certains éléments complexes temporairement pour se concentrer sur la vision d'ensemble du code.
- **Attention !** Le pseudo-code n'a **aucune convention**, mais il existe un certain consensus.
  - https://users.csc.calpoly.edu/~jalbey/SWE/pdl_std.html

L’utilité du pseudo-code

1. Mesurer la difficulté de la mise en œuvre d’un algorithme
2. Mesurer la difficulté de développer une démarche structurée dans la construction d’un algorithme

N.B. 1. Il n’est pas nécessaire de multiplier les éléments de pseudo-code. Il faut se limiter à l’essentiel :

- Lire
- Afficher
- Saisir
- Enregistrer
- *etc*.

N.B. 2. Le pseudo-code peut être représenté sous la forme d’un organigramme.

Les éléments universels (ou presque) dans tout programme

- Les variables
- Les opérateurs
- Les instructions
- Les conditions
- Les boucles
- Les boucles conditionnelles
- Les fonctions / Les procédures
- Les objets

Python

- Le Python est un langage orienté objet à **classes** (1991), inventé par Guido van Rossum (né en 1956).

  Attention ! Le Python est un langage multiparadigme :

  - paradigme de script
  - paradigme impératif
  - paradigme orienté objet
  - paradigme orienté aspect
  - paradigme fonctionnel
  - paradigme orienté logique
  - paradigme de la programmation par contrat

  Remarque. Le nom du langage a été attribué en l'honneur des *Monty Python*.

- Deux versions en activité
  - Python 2
  - Python 3

  Attention ! Le choix est définitif. Il n'existe aucune compatibilité entre les deux versions.

Python

- Langage codé en C, Java et .NET
- Langage gratuit et en licence libre
- Langage interopérable (= portable d'un système d'exploitation à l'autre)
- Langage interprété (≠ langage compilé)
  - Lent
  - Moins performant
- Langage compatible avec d'autres langages
  - C
  - Java
  - *etc*.
- Langage à typage faible (≠ typage fort)
  - Contenu des variables peu importants
- Langage dynamique (≠ langage statique)
  - Plus souple
  - Moins performant

Les instructions

- Une instruction est une séquence identifiée par un début et une fin
  - DÉBUT
    - INSTRUCTIONS…
  - FIN
- En général, les instructions constituent un **bloc**.

La syntaxe dans Python

- Pas d'indicateur de fin de ligne, sauf pour les instructions
  - if, while, for, etc. ont leur ligne de commande se terminant par **:** (**deux points**)
- Indentation obligatoire
  - Retrait marquant un bloc avec **4 espaces**

  N.B. Dans la plupart des I.D.E., dont NotePad++, il suffit simplement d'utiliser une tabulation. L'I.D.E. transforme la tabulation en 4 espaces lors de l'enregistrement du fichier.
- Commentaires :
  - Sur une ligne : **#**
  - Sur plusieurs lignes : **"""… """**

Le choix de l'encodage

- Encodage UTF-8 obligatoire

  #coding: utf8

  – Ne chercher pas à comprendre, il faut mettre ce bout de code à chaque nouveau fichier Python créé

- Indication de la version de Python utilisée (optionnel)

  #! usr/bin/env python3

Les variables

- Une variable représente quelque chose.
- Une variable a un nom (https://dev.to/tontz/comment-nommer-ses-variables-fonctions-et-classes-clean-code-37p6).
- Une variable doit être affectée par un opérateur.
  - L'affectation s'opère de droite à gauche. La variable à gauche subit l'affection, la variable de droite permet l'affectation.
    - X = 1
    - X = X +1
    - Que vaut X ?
- Une variable peut avoir un type :
  - Nombre entier (int)
  - Nombre réel (float)
  - Booléen (0 ou 1 ; OUI ou NON ; VRAI ou FAUX) (bool) : **True** / **False**
  - Chaîne de caractères (texte) (str)

    Attention ! La chaîne de caractères est une variable ayant des spécificités propres qu'il faut approfondir pour n'importe quel langage de programmation.
    - Deux notations possibles :
      - Entre guillemets simples : **'...'**
      - Entre guillemets doubles : **"..."**
  - *etc*.
- Une variable peut être constante.
- Une variable est soit globale (tout le programme), soit locale (spécifique à une instruction).

Les variables en pseudo-code

Déclaration d'une variable

VAR

Nom de la variable : Type de variable

Affectation d'une variable

Nom de la variable ← Valeur de la variable

Les variables en Python

- Toute variable en Python est déclarée par son étiquette :

  var = "Maxime"

  – L'opérateur **=** est l'opération d'affectation.
  – À gauche du =, c'est toujours le nom de la variable.
  – À droite du =, c'est toujours la valeur de la variable.
- Toute constante est en majuscule par convention.

  VAR = input("Bonjour !");

Les variables en Python

- Il est possible de caster (changer leur type affecté dynamiquement) les variables en utilisant les fonctions :

  int(var**)**

  float(var**)**

  str(var**)**

  bool(var**)**
- La fonction **del(**var**)** permet de supprimer une variable.
- Le nom d'une variable doit être **explicite**.
  - Conventions : https://www.gekkode.com/developpement/6-regles-pour-savoir-comment-nommer-les-variables-en-javascript/

Les variables en Python

- Pour tester votre code, vous aurez besoin d'afficher les variables.

  print(*variable*);

- Une fonction pour saisir des données

  saisie = input("Quel est ton nom ?")

Les opérateurs en pseudo-code

- En général, les langages ont :
  - les opérateurs mathématiques classiques
    - +, -, *, /, modulo, division entière, parenthèses, puissance
  - les opérateurs de comparaison
    - ==, != , <, >, <=, >=
  - les opérateurs logiques
    - OU, ET, NON
  - *etc*.

Les opérateurs en Python

- Les opérateurs d’affectation
- Les opérateurs arithmétiques
- Les opérateurs de comparaison
- Les opérateurs de chaîne de caractères
- Les opérateurs logiques

Les opérateurs d'affectation en Python

=	opérateur d'affectation
+=	ajout d'une valeur à une variable affectée
-=	soustraction d'une valeur à une variable affectée
*=	multiplication d'une valeur par une variable affectée
/=	division d'une valeur par une variable affectée
//=	division entière d'une valeur par une variable affectée
%=	modulo d'une valeur et d'une variable affectée
**=	puissance d'une valeur et d'une variable affectée

Les opérateurs arithmétiques en Python

+	addition
-	soustraction
*	multiplication
/	division
//	division entière
%	modulo
**	puissance
(…)	parenthèse (pour les priorités de calcul)

Les opérateurs de comparaison en Python

==	égalité
!=	inégalité
>	strictement supérieur
<	strictement inférieur
>=	supérieur ou égal à
<=	inférieur ou égal à

Les opérateurs de chaîne de caractères en Python

{} concaténation

```
texte = " … {} … {} … "
print(texte.format(variable1, variable2)
```

ou

```
texte = texte1 + texte2
```

== test d'égalité (renvoie un booléen)

!= test de différence (renvoie un booléen)

<, >, >=, <= test de comparaison opéré sur la première lettre de chaque mot testé

Les opérateurs logiques en Python

and	ET
or	OU
not	NON
in	DANS UN ENSEMBLE DE…
not in	EN DEHORS D'UN ENSEMBLE DE…

Les conditions en pseudo-code

- Une condition est une instruction décrivant une alternative pour une structure.
  - SI...
  - ALORS…
  - SINON…
- **N.B.** Il peut exister autant de « SINON » que l'on veut.

Les conditions en Python

```
if var1 == var2:
    instructions si vrai
elif condition:
    instructions si la condition alternative est vraie
else:
    instructions si l'ensemble des conditions sont fausses
```

Les conditions en Python

N.B. Il est possible d'imbriquer des conditions.

Les boucles en pseudo-code

- La boucle est une instruction permettant la répétition d'une structure.
  - RÉPÉTER nombre de fois
    - Instructions…

Les boucles en Python

```
for variable_temporaire in liste:
    instructions
```

Les boucles conditionnelles en pseudo-code

- La boucle conditionnelle est une instruction permettant de répéter une action si une condition est toujours remplie.
  - TANT QUE condition
    - FAIRE instructions…

Les boucles conditionnelles en Python

```
compteur = 0
while compteur < 5:
    instructions
    compteur += 1
```

Cette boucle correspond au pseudo-code « tant que ». Le test définit la condition d'arrêt de la boucle.

Les boucles en Python

N.B. 1. **break** permet d'interrompre la boucle.

N.B. 2. **continue** permet de relancer une itération de la boucle en ignorant le reste des instructions à accomplir.

Les tableaux en pseudo-code

- Les tableaux n'existent pas en Python. Ils sont remplacés par :
  - les listes ;
  - les *tuples* ;
  - les dictionnaires.

Les listes en Python

Une liste est un système de séquences, ou chaque information occupant une place précise.

Une liste est un conteneur.

On travaille directement sur une liste, et non sur une copie comme pour les chaînes de caractères.

Les listes sont très flexibles. Elles peuvent contenir des objets de typage différent (ce qui est généralement interdit dans la plupart des langages de programmation).

La création de listes en Python

1. liste = list()
2. liste = []
3. liste = [1, 3, 15]
4. liste = [0]*10 (liste contenant dix zéros)
5. liste = range(20) (création de la liste : [0, 1, …, 18, 19]

Les listes en Python

Les listes offrent de nombreuses possibilités. Reportez-vous au cours sur GitHub !

Les *tuples* en Python

Un *tuple* est un conteneur.

Un *tuple* est une séquence **immuable**, c'est-à-dire qu'il est impossible de supprimer, d'ajouter des éléments comme dans les listes.

La création de *tuples* en Python

1. tuple = ()
2. tuple = (45,)
3. tuple = 45,
4. tuple = (4, 6)

Les *tuples* en Python

Les *tuples* offrent de nombreuses possibilités. Reportez-vous au cours sur GitHub !

Les dictionnaires en Python

Un dictionnaire est un conteneur.

Un dictionnaire est une liste obéissant à un système de clés-valeurs associées, et non un système indiciel.

Un dictionnaire accepte tous les types.

Un dictionnaire contient des données sans ordre.

La création de dictionnaire en Python

1. dictionnaire = dict()
2. dictionnaire = {}
3. dictionnaire = {clé_1:valeur_1, … }

Les dictionnaires en Python

Les dictionnaires offrent de nombreuses possibilités. Reportez-vous au cours sur GitHub !

Les fonctions en pseudo-code

- Une fonction accomplit une tâche et renvoie une réponse.
  - FONCTION nom de la fonction, paramètres d'entrée de la fonction
    - Instructions…
- Exemple. Élever un nombre au carré
  - FONCTION mettreAuCarre, nombre d'entrée x
    - X = X^2

Les procédures en pseudo-code

- Une procédure accomplit une tâche, mais, à la différence d'une fonction, elle ne renvoie aucune réponse.
  - FONCTION direBonjour, nom de l'utilisateur
    - Afficher "Bonjour + nom de l'utilisateur !"

Les fonctions ou procédures

- Pas de panique ! Python ne distingue pas les deux.
- Il existe de nombreuses fonctions natives :
  - fonctions mathématiques
  - fonctions de conversion des variables
  - fonctions sur les chaînes de caractères
  - fonctions sur les listes, *tuples* et dictionnaires
  - *etc*.

Les fonctions en Python

- Une **fonction** affecte un bloc d'instructions qui ne fait qu'**un** traitement.
  - Elle a un identificateur.
  - Elle peut avoir des paramètres.

```
def nomDeLaFonction(paramètre 1, paramètre
  2, …):
   instructions
```

Les fonctions en Python

```
def f(x)
{
    return x*x;
}
carreDeDeux = f(2) ←
```

*appel de la fonction f avec l'**argument** 2*.

return en Python

return est la commande qui permet de renvoyer une valeur une fonction que la fonction à terminer son exécution.

N.B. Si aucun return n'est pas placé en fin de fonction, il existe une valeur de retour est par défaut (à regarder dans la documentation).

Les fonctions en Python

- Les fonctions sont **en principe** définies **au début** du programme.

N.B. 1. Lorsque Python rencontre une fonction, il ne l'exécute pas directement ; il enregistre le bloc d'instructions sous le nom de la fonction.

N.B. 2. Une fonction peut utiliser les variables d'une portée globale.

N.B. 3. Il est possible de faire une **fonction anonyme** (= fonction lambda).

- Les fonctions natives ou créées offrent de nombreuses possibilités. Reportez-vous au cours sur GitHub !

Les objets en pseudo-code

- Un objet est un moyen de structurer les variables (ou données).
- Un objet contient deux groupes d'éléments :
  - des propriétés, des attributs (les variables de l'objet)
  - des méthodes (les fonctions de l'objet)
- Sauf exceptions, pour utiliser un objet, il faut le créer (en paramétrant ses propriétés) et accéder à ses méthodes.

- Python est un langage orienté objet. L'objectif de l'algorithmique est de comprendre les instructions élémentaires afin de pouvoir concevoir des objets.

Les objets en Python

- La notion d'objet en Python est très complexe, et parfois ambiguë.

```
class Image(source, titre, largeur, hauteur)
   variableDeClasse1 = …
   variableDeClasse2 = …
   def __init__(self):
      self.source = source;
      self.titre = titre;
      self.largeur = largeur;
      self.hauteur = hauteur;
   def methode1():
       instructions
   def methode2(parametre1, parametre2):
       instructions

photo = new Image("photo.jpg", "Ma photo", 80, 120);
```

Appel d'une méthode :

```
photo.methode1()
photo.methode2(1, 2)
```

N.B. self est utilisé pour renvoyer à l'objet lui-même, c'est-à-dire ici Image.

Les objets en Python

- Le Python ne respecte pas le principe d'encapsulation. De fait, vous pouvez appeler les attributs de l'objet, ce qui est interdit dans la plupart des langages.
- Python ne respecte aucun principe informatique concernant le concept d'héritage.

N.B. En Python, les objets sont des dictionnaires qui définissent les attributs (particularité du langage).

Les objets en Python

- Tout est objet en Python. Une fois cette idée est bien comprise, il est possible d'approfondir les objets natifs.
- Tous les objets que vous utiliserez seront contenus dans des bibliothèques
- À vous de découvrir toutes les possibilité du Python !

Les bibliothèques en Python

- Une bibliothèque est une collection de fonctions, de classes, de constantes ou autres variables. Elle permet d'organiser et de hiérarchiser cette collection.
- Une bibliothèque en Python est un module.
- Un module est un fichier ***.py**.

Exemple : le module de mathématique

```
import math
math.sqrt(2)
```

math.sqrt(2) utilise l'objet **math** et appelle sa méthode **sqrt** (racine carrée).

La modularité en Python

- Lorsque vous serez aguerri, vous pourrez créer vos propres modules avec Python, en attendant on utilisera ceux des autres :
  - matplotbit
  - seaborn
  - *etc*.
- À vous de découvrir toutes les possibilité du Python !