

NSI 1ère - Algorithmique - 3 - Pseudo langage, Python

QK

Pseudo langage

Notions de pseudo langage

- langage formel minimal pour décrire un algorithme
- un langage de programmation (Python, java, C) est trop contraignant
- dans les livres les algos. sont écrits en pseudo langage.

Éléments communs

Tous recouvrent les mêmes concepts :

- variables, affectations
- structure de contrôle : séquence, condition, itération
- découpage de l'algo. en sous-programmes (fonctions)
- structures de données (tableaux, dictionnaires etc.)

Variables, affectations

- Les variables sont indiquées avec leur type : booléen b, entier n etc.
- on affecte avec = ou \leftarrow \leftarrow illustre bien l'affectation : “mettre dedans.”

Structures de données

- Les tableaux sont utilisés. Si A est un tableau, $A[i]$ est le i^{eme} élément de ce tableau.
- Les structures sont utilisées. Si P est une structure modélisant un point et x un champ modélisant l'abscisse alors $P.x$ est l'abscisse de ce point.

Séquencement des instructions

- si nécessaire, on termine une instruction avec ;
- Les blocs d'instructions sont entourés de
 - {...}
 - début ... fin
- En Python, ce n'est nécessaire que sur un même ligne.

```
a = 3; print(a)
```

Conditionnelle

La conditionnelle est donnée par :

```
si (condition){  
    instruction1;  
}sinon{  
    instruction2;  
}
```


Python et l'indentation

Indenter :

Mettre des espaces en début de ligne

- Pseudo code, langage avec syntaxe inspirée du C :
La structure est indiquée par les { }
indentation optionnelle, pour éclairer le lecteur
- Python :
Dans Python la **structure est donnée par l'indentation**
Elle est **OBLIGATOIRE**

Conditionnelle en Python

```
a = 5
if a > 4: # apres : on indente
    print("plus grand que 4") # dans le bloc if
else:
    print("inférieur ou égal à 4") # dans le bloc else
print("le if est terminé") # sera exécuté
```

qui affiche :

```
plus grand que 4
le if est terminé
```

Python : if, elif, else

a % 3 le reste dans la division par 3 de a

% se lit "modulo"

```
a = 16
if a % 3 == 0: # si a est divisible par 3
    print("divisible par 3, le reste vaut 0")
elif a % 3 == 1:
    print("le reste vaut 1")
else:
    print("le reste vaut 2")
```

Itérations

Nous utiliserons plusieurs types de boucles :

Les boucles while

tant **que** (condition){ faire **ceci...** }

En Python :

```
n = 0
while n < 5:
    print(2 * n + 1)
    n += 1
1, 3, 5, 7
```

Contexte d'utilisation : `while`

On emploie principalement les boucles `while` quand **on ne sait pas combien d'étapes seront nécessaires**.

Itérations

Les boucles for

pour i allant de min à max { faire ceci }

Et en Python :

```
for i in range(5):  
    print( 2 * i + 1)
```

Itérer dans une liste en Python.

```
for mot in ["une", "liste"]:  
    print(mot)
```

une

liste

Rq : Il est possible d'itérer sur de nombreux objets dans Python :

liste, tuples, dictionnaires, set, fichiers, générateurs etc.

Ici on **itère** sur la liste,

i référence successivement ses éléments.

Contexte d'utilisation : for

- Quand on sait combien d'étapes seront nécessaires
- Quand on veut parcourir tous les éléments d'un "paquet" (liste, tableau, dictionnaire etc.).

Fonctions

- Une fonction est un “petit” programme qui renvoie une valeur.
- Elles permettent
 - un découpage qui **facilite la compréhension**.
 - de **factoriser** : on évite ainsi d’écrire plusieurs fois la même série d’instruction.

Fonctions en Python

Elles sont définies avec **def** En sortie, elles renvoient des variables avec **return**

```
def carre(x):  
    return x ** 2  
  
a = carre(9) # 81
```

Fonction sans sortie

```
def direBonjour(texte):  
    print("Bonjour je m'appelle {}".format(texte) )
```

```
direBonjour("Henri")
```

Bonjour je m'appelle Henri

Rq : `.format(...)` est une **méthode** de la classe **str**

Intérêt des fonctions :

Les fonctions facilitent le développement :

Il est plus facile de trouver une erreur parmi 10 fonctions de 3 lignes que dans un bloc de 30 lignes.

Portée des variables

On distingue les variables **locales** et **globales**

```
a = 5 # variable globale
def maFonction():
    a = 3 # variable locale à la fonction
    print(a)
maFonction()
print(a) # a vaut 5 !!!
```

3

5

En dehors des fonctions : variables **globales**

Dans les fonctions : variables **locales**

Attention aux arnaques : Python Tutor - portée des variables

Python : utiliser une variable globale

```
a = 5 # variable globale
def maFonction():
    global a # on rend la variable globale
    a = 3
maFonction()
print(a)
3
```

Cette fois, on a spécifié `global a` !!

En pseudo code

On évite cette difficulté en précisant les paramètres (entrées et sorties) de la fonction.

- `maFonction(e int i, s int j, es int k)`
- en entrée : avec **e** on passe à la fonction la valeur **i** mais elle ne la changera pas globalement
- en sortie : la fonction écrit dans **j** mais elle n'en connaît pas la valeur extérieure
- en entrée/sortie : on lui passe la valeur de **k** et elle écrit dedans.
- Sans précision, on suppose que c'est en entrée.
- passage par entrée/sortie = **passage par référence** ou **passage par variable**

Exemple

- code appelant :

$i \leftarrow 3$

$j \leftarrow 5$

$k \leftarrow 8$

maFonction(e int i, s int j, es int k);

Afficher(i, j, k);

- maFonction(e int i, s int j, es int k){

$i \leftarrow i + 1$

$j \leftarrow 6$

$k \leftarrow k + 2 \}$

- résultat de Afficher(i, j, k) : ? ? ?

Exemple - solution

- code appellant :

$i \leftarrow 3$

$j \leftarrow 5$

$k \leftarrow 8$

maFonction(e int i, s int j, es int k);

Afficher(i, j, k);

- maFonction(e int i, s int j, es int k){

$i \leftarrow i + 1$

$j \leftarrow 6$

$k \leftarrow k + 2 \}$

- résultat de Afficher(i, j, k) : 3 6 10

fonctions : différentes implémentations

- En Java certains objets n'ont de paramètres qu'en entrée (int, float etc.)

Python : prudence !

- En Python il faut faire attention, à nouveau...

```
def f(x):  
    x = 5  
a = 0  
print(a) # 0  
f(a)  
print(a) # 0
```

a été passé en entrée

Tandis que...

```
def f(l):  
    l[0] = 5  
a = [0, 1, 2]  
print(a[0]) # 0  
f(a)  
print(a[0]) # 5
```

Cette fois on modifie l'élément 0 de la liste a !