Le cours d'algorithmique

Formation d'ingénieur de l'université des Antilles

Auteur : Vincent Pagé : vincent.page@univ-antilles.fr

Introduction

Dans ce cours, nous verrons les rudiments de l'algorithmique pour vos formations (Matériaux et Systèmes énergétiques). Mon objectif est simple : ne pas focaliser sur les détails, mais vous permettre de faire des choses rapidement. Ce document est extrêment synthétique. Il ne vous dispense pas d'aller en cours ni d'essayer de programmer vous même. Pour plus de détails, vous pouvez me poser des questions et/ou chercher un peu sur le net ou dans des cours plus détaillés.

Les concepts présentés s'appliquent à presque tous les langages que je connais. Lorsque je devrais faire un vrai programme pour vous présenter quelque chose, je le ferais en python. Installez donc python sur votre machine. Les exemples seront fait pour fonctionner en python 3.

Le dépot dans lequel vous avez trouvé ce document contient les exemples que nous avons vu en cours.

Le cours d'algorithmique

Formation d'ingénieur de l'université des Antilles

Auteur : Vincent Pagé : vincent.page@univ-antilles.fr

Cours 1

Concepts d'algorithmique d'aujourd'hui.

Ceci est ma vision de la programmation actuelle. Tout le monde ne s'y retrouvera peut être pas, mais il me semble que la plupart des développeurs en entreprise seront d'accord avec le constat suivant : Programmer (ou coder) fait appel à deux grands capacités : - la stratégie - la tactique

Explications : TODO

un cours focalise souvent sur la tactique, alors que les langages actuels permettent d'en faire abstraction assez fréquemmment. Par exemple, si je dois trier un tableau par ordre croissant, nous avons, dans tous les langages évolués, des fonctions pour le faire.

La stratégie, elle est utile pour programmer mais aussi pour la coordonnation de n'importe quel projet que vous aurez à réaliser.

Voyons donc le minimum de tactique à savoir pour commencer

Les variables

Comprenons une chose tout d'abord : un programme informatique ne fait qu'une chose : il manipule des variables. Toutes les informations que doit gérer votre programme doivent donc se retrouver dans des variables. Vous en connaissez vraisemblablement quelques types simples de variables : - les entiers (int) - les nombres à virgules (float) - les vrais ou faux (booleen) - les chaînes de caractères (string)

Dans les variables, je stocke des valeurs.

```
a=5
b=7
print(a+b)
```

j'ai créé une variable a, lui ai donné la valeur 5, puis crée une variable b, lui ai donné la valeur 7, puis affiché le résultat de la somme des valeurs des deux variables.

Les tests (if)

TO DO

Les boucles Tant que (while)

Les fonctions.

Le code que j'ai présenté juste avant est le programme principal. C'est ce que fait mon programme.

Un vrai bon programme découpe le code en petites actions que le programme principal organise. Si je veux programmer ## Cours 2 : Tableaux

intérêt

Avec les types de variables que nous avons vus dans le cours précédent, nous pouvons faire beaucoup de choses, mais cela devient vite pénible.

Imaginons que je doive faire un programme qui gère vos notes, je doiS stocker une note (un float) par étudiant. Je pourrait par exemple choisir un ordre pour rentrer les notes et stocker la note de chaque étudiant dans une variable. Disons que ma classe ne contient que 3 étudiants.

```
e1 = 12
e2 = 9.5
e3 = 14
```

Pour calculer la moyenne de ma promotion, je pourrais ajouter ceci au code qui précède :

```
moyenne = (e1+e2+e3) / 3
print (moyenne)
```

Si finalement, je dois ajouter un 4eme étudiant arrivé en retard, mon code est transformé à de multiples endroits pour devenir :

```
e1 = 12
e2 = 9.5
e3 = 14
e4 = 2 # oui, il est mauvais, en plus

moyenne = (e1+e2+e3+e4) / 4
print (moyenne)
```

A chaque ajout d'étudiant, il faudra que je pense à faire toutes les modifications. A terme, je suis sûr de faire une erreur...

De plus, dans le cas de votre promotion, il me faudrait une vingtaine de variables, ce qui devient lourd et pénible. Il est temp d'introduire les tableaux. En python, on les appelle des *listes*, mais c'est sans importance pour nous.

Voici ce que je voudrais : une seule variable contenant ces informations, rangées dans des cases séparées.

notes
12
9.5
14
2

Ici, *notes* est le nom de mon tableau. L'intérêt est simple : si j'ajoute un étudiant, j'ajoute juste une case à mon tableau. **Mes notes ne sont contenues que**

dans une seule variable : *notes*

Je peux créer un tableau de ce type comme ceci

```
notes = [12, 9.5, 14, 2]
print(notes)
```

Pour accéder à une case, je vais utiliser son numéro (on parle d'*indice* de la case). Les indices commencent à 0. Une bonne représentation de mon tableau serait la suivante :

indice	valeur
0	12
1	9.5
2	14
3	2

Si je veux accéder à la case numéro 2, j'utiliserais l'écriture suivante : notes[2]

Le code suivant : - affiche la valeur de la case 2 (qui vaut 14) - modifie la valeur de la case 3 pour y mettre la valeur 11 - affiche tout le tableau

```
print (notes [2])
notes[3] = 11
print(notes)
```

Manipulations de base sur les tableaux

Création d'un tableau

Pour créer un tableau, on peut le créer déja rempli, comme nous l'avons fait. Nous aurions pu également créer un tableau vide et le remplir quand nous voulons (ce qui permettrait d'ajouter des étudiants à n'importe quel moment)

Ce qui suit crée un tableau vide, et le remplit avec des chaines de caractères contenant les noms de chaque étudiant de ma promo.

```
noms = []

print (noms)

noms.append("moutoussamy")
noms.append("destouches")
print (noms)

noms.append("julan")
print (noms)
```

```
noms.append("najeus")
print (noms)
```

longueur d'un tableau

il peut être utile de connaitre le nombre de cases d'un tableau nommé *tab* : on l'obtient avec la fonction *len*

```
nbEtudiants = len(noms)
print (nbEtudiants)
```

parcours de tableaux

Si je veux afficher le contenu de chaque case du tableau de notes, je peux utiliser print(notes). Ici, je vais le faire d'une autre manière, que je réutiliserais de plusieurs façons différentes utilisant toutes la notion de parcours de tableau. Cela me permettra de faire des choses plus compliquées plus tard (comme calculer la moyenne ou trouver le nom du major de promo).

Si je veux afficher le tableau, ce que je veux faire est en fait afficher successivement le contenu de chaque case.

Je vais donc visiter chaque case du tableau afficher le contenu de la case en cours.

Nous allons voir 3 façon de le faire, chacune ayant ses intérêts (surtout les 2 dernières en fait)

parcours d'un tableau avec une boucle while.

je peux le faire comme suit avec la boucle *while* vue dans le cours précédent :

```
notes = [12, 9.5, 14, 2]

i = 0
while i<4 :
    print(notes[i])
    i = i+1
```

Mais ceci ne marche que pour un tableau de 4 cases. Je peux améliorer ceci facilement en modifiant légèrement mon code en prenant en compte la taille du tableau.

```
notes = [12, 9.5, 14, 2]

i = 0
while i<len(notes) :
  print(notes[i])
  i = i+1
```

C'est fonctionnel mais peu sympathique à écrire. Voyons une version plus pratique.

parcours d'un tableau avec une boucle for.

```
notes = [12, 9.5, 14, 2]

for e in notes :
  print (e)
```

Dans ce type de boucle, à chaque tour de boucle, la variable *e* va prendre la valeur du contenu de la case visitée. le *for* se débrouille tout seul pour se promener de case en case.

Le code précédent se lit quasiment comme en francais : pour chaque *e* dans le tableau *notes*, j'affiche *e*

Vous choisissez le nom que vous donnez à la variable qui visite les cases. Le nom du tableau (après *in*) est celui que vous avez donné à votre variable contenant le tableau. Je pourrais tout aussi bien écrire :

```
notes = [12, 9.5, 14, 2]

for biten in notes :
  print (biten)
```

C'est la version la plus simple pour parcourir tout tableau.

parcours avec une boucle for et l'indice de la case

il peut arriver que j'ai besoin de me déplacer dans mon tableau en utilisant le numéro des cases du tableau. C'est ce que nous allons essayer de faire ici...

Commençons par ce code. Vous devriez vite comprendre qu'il affiche les chiffres de 0 à 3.

```
indices = [0,1,2,3]

for i in indices :
  print (i)
```

Je peux me servir de ce *i* variable pour afficher le contenu de la case numéro *i* d'un tableau, comme ceci :

```
for i in indices :
  print (notes[i])
```

Le problème est que je dois définir manuellement le tableau indice. Ce qui est pénible, si mon tableau a de nombreuses cases. Pour générer un tableau allant de 0 à n-1, nous disposons de *range(n)*.

```
indices  = range(4)
for i in indices :
  print (notes[i])
```

ou encore

```
for i in range(4) :
  print (notes[i])
```

Mais ceci ne fonctionne que pour des tableaux de 4 cases. Il suffit d'intégrer la longueur du tableau à mon code et c'est réglé :

```
for i in range(len(notes)) :
  print (notes[i])
```

Quelques exemples

Si vous avez compris ces parcours, au lieu d'afficher simplement le contenu de chaque case, nous allons pouvoir faire des choses plus complexes.

Somme des éléments d'un tableaux

Pour faire la somme des éléments d'un tableau, il me suffit d'avoir une variable *somme* qui vaudra 0 au départ, et que je vais augmenter au cours de mon parcours de la valeur de chaque case visitée.

N'importe quel parcours parmi les 3 précédents fonctionne, je vais prendre le plus simple.

```
notes = [12, 9.5, 14, 2]

somme = 0
for n in notes :
    somme = somme + n

print ("somme :", somme)
```

A moindre frais, je peux aussi calculer la moyenne en ajoutant la ligne qui suit :

```
print ("moyenne :", somme/len(notes))
```

recherche du maximum d'un tableaux

Si je cherche le maximum de mon tableau, il suffit que je dispose d'une variable *maxi* qui vaut, disons 0. Je parcours mon tableau et chaque fois que je vois quelque chose de plus grand que ce que j'ai déja vu, mon *maxi* est mis à jour.

Quelque chose comme ceci.

```
notes = [12, 9.5, 14, 2]

maxi = 0
for n in notes :
    if n > maxi:
      maxi =  n

print ("somme :", somme)
```

Avec un peu d'expérience, je peux me méfier de la ligne qui dit *maxi = 0*. En effet si mon tableau ne contient que des éléments négatifs, aucune case ne va déclencher mon *if*. Par conséquent, a la fin de la boucle, le maximum de mon tableau sera resté à zéro, alors que tous les éléments sont négatifs... De fait, il vaut mieux initialiser mon maxi à la valeur de la premiere case du tableau, comme suit :

maxi = notes[0] for n in notes : if n > maxi: maxi = n

print ("maxi :", maxi)

parcours de 2 tableaux conjoints

Bon. Imaginons que je veuille maintenant gérer aussi les noms de mes étudiants. Je pourrais stocker le nom des étudiants dans un tableau de chaînes de caractères.

```
noms = ["moutoussamy","destouches","julan","najeus"]
```

L'idée est que l'étudiant dont le nom est dans la case numéro 2 du tableau *nom* a eu la note contenue dans la case numéro 2 du tableau *notes*

Si je souhaite afficher les noms et les notes de ma promotion : je vais me déplacer de case en case en utilisant un indice variant de 0 à 3. Pour chaque indice, j'affiche la case correspondante dans le tableau des noms et la case correspondante dans le tableau de notes.

```
noms = ["moutoussamy","destouches","julan","najeus"]
for i in range(len(notes)):
    print(noms[i], notes[i])
```

Cela commence a ressembler a quelque chose ! Voyons si l'on peut compliquer un tout petit peu.

affichage du nom du major

Pour afficher le nom du major de promo, c'est relativement simple : je vais chercher le numéro de la case contenant le maximum du tableau de notes. Je peux alors afficher le nom situé dans la case correspondante dans le tableau de noms.

Pour trouver la position du maximum : on veut retenir la valeur du maximum, mais aussi sa position. On aura donc ces variables à mémoriser.

Lors du parcours du tableau, chaque fois que je mets a jour mon maximum, je met également à jour sa position.

```
maxi = notes[0]
imaxi = 0

for i in range(len(notes)):
    if notes[i]> maxi:
        maxi = notes[i]
        imaxi = i

print ("major", noms[imaxi])
```

Nettoyage et mise en fonctions.

Pour avoir du code propre il est toujours recommandé de coder en utilisant des fonctions.

Chacune des petites actions que j'ai faites va donc être déplacée dans une fonction spécifique. A dire vrai, j'aurais tendance à le faire au fur et à mesure. Ici, et c'est la dernière fois de l'année, je le fais à la fin du chapitre pour ne pas tout mélanger.

En se souvenant de ce qui a été fait dans le cours précédent sur les fonctions, vous devriez pouvoir comprendre ce qui suit, qui reprend la totalité de nos travaux, mais proprement.

```
def afficher(tab):
    for e in tab :
        print(e)

def afficherPromo(noms, notes):
    for i in range(len(noms)) :
        print(noms[i], notes[i])

def calculerSomme(tab):
    somme = 0
    for e in tab :
        somme = somme + e

    return somme

def maximum(tab):
    maxi = tab[0]
    for e in tab :
```

```
        if e > maxi:
            maxi = e

    return maxi

def imaximum(tab):
    maxi = tab[0]
    imaxi = 0

    for i in range(len(tab)) :
        if tab[i] > maxi:
            maxi = tab[i]
            imaxi = i

    return imaxi

notes = [12, 9.5, 14, 2]
noms = ["moutoussamy","destouches","julan","najeus"]

afficher(noms)
afficher(notes)

afficherPromo(noms, notes)

somme = calculerSomme (notes)
print ("somme :", somme)

moyenne = somme / len(notes)
print ("moyenne :", moyenne)

maxi = maximum(notes)
print ("maxi :", maxi)

iMajor = imaximum(notes)

print ("major :", noms[iMajor])
```

les fichiers

Les sources de tout ce que nous avons fait dans ce cours sont dans le répertoire Sources. On y trouvera en particulier : - L'intro sur les tableaux - Les parcours de tableaux - Les exemples de calcul - La version finale ## Sommaire - Introduction - Cours 1 - Cours 2

Etat du cours :

Pour le moment : - le cours 1 n'est pas finalisé - le cours 2 est propre

Cours Complet

Il existe aussi une version complète du cours qui réunit tous les cours - format md - format pdf