Travaux pratiques en Python

Table des matières

Séances de programmation en Python 3 : prise en main, mise en œuvre des cours d'algorithmique, manipulation de la tortue Python, recherche dans les listes, etc.

I. Algorithmique et Python

A. Exercice 1 : Les basiques de Python 3

Dans cette séance, on s'approprie l'environnement : l'éditeur de texte où l'on tape les programmes, l'enregistrement des programmes, l'appel à l'interprète Python dans le terminal.

1. Affichages

1. Tester l'instruction d'affichage `print` de Python : afficher un nombre, une chaîne de caractères, une variable.
2. Comment peut-on passer ou ne pas passer à la ligne ?

2. Boucles

1. Écrire une boucle `while` qui compte jusqu'à 100. Idem avec une boucle `for`.
2. À l'aide de boucles imbriquées, dessiner des figures géométriques : triangles et carrés, creux ou non. Idem à l'aide de procédures et de leurs paramètres.

3. Listes

1. Créer une liste vide. Créer une liste contenant des éléments.
2. Afficher la longueur de cette liste.
3. Remplacer l'une des valeurs, enlever une case et en ajouter une autre, puis afficher à nouveau la liste.
4. Utiliser une boucle pour parcourir les éléments de la liste et les afficher un par un. Passer en revue les différentes syntaxes possibles.

4. Aléatoire en Python

1. Tester les fonctions de la librairie `random` suivantes :
   `random.randint`, `random.randrange`, `random.choice`, `random.shuffle`.

B. Exercice 2 : tableaux de nombres

1. Définir en Python un tableau d'entiers.

a) Écrire (et valider sur le tableau déjà défini) les fonctions qui :

1. Calcule la moyenne des nombres contenus dans un tableau donné,
2. Compte le nombre d'occurrences d'un élément,
3. Compte combien d'éléments sont supérieurs ou égaux à 10,
4. Recherche la valeur maximale du tableau,
5. Teste si un élément est présent ou non, etc.

Nous nous intéressons maintenant à des tableaux beaucoup plus grands. Tout d'abord nous voulons créer de tels tableaux remplis de manière aléatoire, à l'aide du module `random`.

b) Écrire les fonctions qui, pour une taille donnée *n* :

1. Fournit un tableau de *n* entiers aléatoires,
2. Construit le tableau des *n* premiers entiers mélangés aléatoirement.

Enfin, nous utilisons le module `time` pour mesurer le temps nécessaire à l'exécution d'un bloc d'instructions.

1. Sur des grands tableaux, mesurer le temps nécessaire à chacune des fonctions calculant la moyenne et recherchant un élément.

C. Exercice 3 : tableaux d'enregistrements, les étudiants

1. Créer une promotion comme un tableau d'étudiants,
2. Implémenter la fonction *moyenne d'un étudiant* dédiée cette représentation,
3. Pour chaque discipline, la fonction *moyenne de la promotion*,
4. La fonction qui trouve l'étudiant ayant eu la note moyenne maximale, etc.
5. Généraliser à un type d'étudiant qui possède un nombre quelconque de notes.

D. Exercice 4 : Algorithmes avancés sur les tableaux

1. Sur les tableaux quelconques

1. Implémenter une fonction `index_minimum(t,d,f)` qui renvoie le numéro de la case contenant la plus petite valeur du tableau `t` entre les cases `d` et `f`.
2. Programmer un *tri à bulles*.

2. Sur les tableaux déjà triés

On suppose disposer d'un tableau de nombres rangés par ordre croissant.

1. Implémenter une fonction de *recherche* d'un élément utilisant une boucle `tant que` et tirant parti du fait que les éléments sont ordonnés.
2. Écrire une fonction de *recherche dichotomique*.
3. Proposer une procédure `insertion(e,t,n)` qui ajoute un élément `e` à sa place dans un tableau `t` de taille `n`.

3. Autres méthodes de tri

1. `tri_extraction` utilisant `index_minimum(t,d,f)` : on récupère le minimum du tableau et on le place dans la première case, on récupère le minimum du tableau privé de la première case et on le place dans la deuxième, etc.
2. `tri_insertion` utilisant `insertion(e,t,n)` : prendre le $i^{ème}$ élément et le mettre à sa place dans les *i-1* premières cases déjà triées.

E. Exercice 5 : Comparaison expérimentale des méthodes de tri

1. Préliminaires

On veut dans cette séance comparer les méthodes de tri (comme le *tri à bulles* par exemple) en termes de temps de calcul et en fonction de la taille et de la nature des tableaux à trier.

De nouveaux points techniques sont nécessaires, essentiellement : savoir produire des fichiers texte en Python et connaître les bases de `gnuplot`.

`gnuplot` est un outil qui permet de tracer des courbes à partir de données brutes. Par exemple, l'instruction :

$ gnuplot

gnuplot> plot 'stats.dat' with lines

Permet de tracer les points qui se trouvent dans le fichier `stats.dat (à créer)` sous la forme suivante (un point par ligne, valeur en abscisse, une tabulation, valeur en ordonnée) :

```
100     2
200     14
300     26
400     48
500     72
600     111
700     142
800     194
900     238
1000    298
```

C'est ce type de fichier que l'on veut faire produire par Python (la première colonne pourrait correspondre aux tailles des tableaux considérés, la seconde aux temps de traitement nécessaires).

2. Fonctions à implémenter

1. Écrire une fonction `copie (t)` qui renvoie un nouveau tableau contenant dans le même ordre les mêmes valeurs que le tableau `t` ; vérifier qu'une modification de la copie n'altère pas le tableau original.
2. Proposer une fonction `inverse (t)` qui fournit un nouveau tableau contenant les mêmes valeurs que le tableau `t` mais dans l'ordre inverse.
3. Implémenter des fonctions pour produire des tableaux :
   - Une fonction `tableau_premiers_entiers (n)` qui produit un tableau contenant dans l'ordre tous les entiers de 1 à `n`,
   - Une fonction `tableau_premiers_entiers_melanges (n)` qui propose ces mêmes entiers mélangés aléaoirement,
   - Une fonction `tableau_premiers_entiers_inverses (n)` qui propose ces mêmes entiers du plus grand au plus petit.

4. Proposer une procédure `ligne_dans_fichier (f,n,t)` dont le rôle est d'écrire dans le fichier `f` la valeur (numérique) de `n`, une tabulation, la valeur (numérique) de `t` et enfin un passage à la ligne.
5. Écrire une fonction `temps_tri_bulles (t)` qui fait une copie du tableau `t` et renvoie le temps nécessaire au tri à bulles pour classer cette copie.
6. Coder la procédure `stats_melange (nmin,nmax,pas,fois)` qui pour chaque taille de tableau comprise entre `nmin` et `nmax` en avançant de `pas` en `pas` produit `fois` tableaux mélangés aléatoirement et écrit dans un fichier le temps moyen nécessaire au tri à bulles pour classer ces tableaux.
7. Même question avec la fonction `stats_ordonne (nmin,nmax,pas,fois)` pour des tableaux déjà ordonnés.
8. Même question avec la fonction `stats_inverse (nmin,nmax,pas,fois)` pour des tableaux déjà ordonnés mais en ordre inverse.
9. Produire à l'aide de votre code des fichiers de statistiques pour des tailles de tableau variant de 100 en 100, entre 100 et 1000, avec 5 répétitions pour chaque taille de tableau. Visualiser ces données avec `gnuplot` et comparer l'évolution du temps nécessaire au tri bulles selon le type de tableaux et selon leurs tailles.
10. Généraliser votre code pour pouvoir également comparer les méthodes de tri entre elles : tri à bulles, tri insertion, tri extraction et tri rapide.
11. Tester des modifications des méthodes de tri, calculer les écarts-types des temps de calcul sur les tableaux *mélangés*, explorer les possibilités de `gnuplot`, expliquer théoriquement les courbes obtenues.

Au final, on doit obtenir des images comme celles-ci :