TP 8 VARI 1

Démarrer une console/terminal

Rappel : `man` ***xyz*** affiche le manuel d'une commande ***xyz*** (remplacer ***xyz*** par votre vraie commande). Taper `/abc` et *Entrée* pour chercher `abc` dans le manuel. Pour quitter le manuel, taper "q".

Exercice 1 Taper `man du` pour trouver le manuel de la commande `du`.

- Quel est l'objectif de cette commande `du` ?
- À quoi sert l'option `--max-depth` ?

Exercice 2 Taper la commande suivante pour afficher l'espace disque utilisé par chaque dossier dans le dossier courant

```
du  -h  --max-depth=1
```

Que se passe si on remplace `max-depth=1` avec `du -h --max-depth=2` ?

Quel est le résultat de la commande suivante ?

```
du  -h  --max-depth=1|sort -h
```

Exercice 3 Taper la commande suivante pour créer un dossier `tp8vari1` pour y mettre vos programmes.

```
mkdir   tp8vari1
```

Se placer dans ce dossier :

```
cd   tp8vari1
```

Par la suite taper

```
echo "salut"> toto.txt
```

pour écrire « salut » dans un fichier `toto.txt`. Visualiser ce fichier à l'aide d'une commande `cat`. Ouvrir le gestionnaire de fichiers et vérifier que vous trouvez le texte « salut » dans ce fichier.

Programmation : vous pouvez choisir de faire les exos non graphiques sous Java ou Processing

Exercice 1 Écrire une fonction `doubler(int)` qui renvoie le double d'un nombre entier. Remplir le code ci-après pour le faire fonctionner ; vous pouvez choisir de travailler soit sous `processing` soit sous `java`.

PROCESSING :

```
... doubler (int x){
    ....
}
void setup(){
   int salaire = 1000;
   int nouveauSalaire = doubler(salaire);
   println("J'ai doublé le salaire et mon nouveau salaire est "+nouveauSalaire);
}
```

JAVA :

```
class Exo1{
   static int doubler (int x){
   ....
   }
   public static void main(String[] args){
      int salaire = 1000;
      int nouveauSalaire = doubler(salaire);
      ....println("J'ai doublé le salaire et mon nouveau salaire est "+
         nouveauSalaire);
   }
}
```

Exercice 2 Écrire une fonction qui renvoie le cube d'un `float`. Ce programme devrait afficher 1030.301.

```
... cube (...){
    ....
}
void setup(){//ou main(..) sous java
   float lecube=cube(10.1);
   println(lecube);
}
```

Exercice 3 Remplir la fonction `calcIntérêts` ci-dessous pour calculer les intérêts gagnés au taux `t` sur un capital `cap`. Par exemple, `calcIntérêts(1000,0.05)` devrait renvoyer 50. Remplir le programme ci-après.

```
... calcInterets (float cap, float t){
    ....
    return ...
}
void setup(){
  float x ;
  x = calcInterets(1000,0.05);

  println("Capital final =" ...);
}
```

Exercice 4 Soit le programme ci-après. Dire ce qu'il affiche sans le faire tourner. Dans une deuxième étape, taper le code et vérifier votre réponse.

```
int detMax(int a, int b){
    if(a>b)
        return a;
    return b;
}
void setup(){
    int a=9, b=8;
    println(detMax(b,5));
    println(detMax(5,a));
}
```

Exercice 5 Écrire une fonction `detMaxTab(int[])` qui appelle 2 fois la fonction `detMax(...)` ci-dessus pour trouver le maximum d'un tableau de 3 cases. Écrire un fonction similaire `detMinTab(int[])` qui détermine le minimum. Ajouter une fonction `notesValides(int[])` qui renvoie `true` si les notes d'un tableau de 3 cases sont comprises entre 1 et 20 et `false` sinon. Commencer avec :

```
boolean notesValides(int[] notes){
    if((detMinTab(notes)>0)&& ...)
        return true;
    ... ... ...
```

Exercice 6 Le programme ci-après permet de tracer des lignes à des positions aléatoires. **Rappel :** La fonction `draw()` est appelée de manière répétitive en continu. L'appel de fonction `random(600)` renvoie un `float` aléatoire entre 0 et 600. Modifier le programme pour le faire afficher des ellipses de couleurs différentes (utiliser `fill(random(255),...,...)`). Les tailles des ellipses doivent être inférieures à 300.

```
void setup(){
    size(600,600);
}
void draw(){
    line(random(600),random(600), //1er pixel ligne: coordonnées aléatoires <600
         random(600),random(600));//2ème pixel ligne:coordonnées aléatoires <600
}
```

Exercice 7 **(bonus)** Écrivez un programme qui commence par dessiner 5 cercles rouges de rayon 50 à des positions aléatoires sur une toile de taille 500 × 500. Cette tâche doit être exécutée dans la fonction `setup()`. Chaque appel à la fonction `draw()` doit essayer de dessiner un cercle vert de rayon 50 à une position aléatoire sur la toile. En fait, la condition suivante doit être satisfaite : le cercle vert ne doit pas intersecter l'un des cercles rouges initiaux. Si c'est le cas, l'appel courant à `draw` doit simplement ne pas dessiner le cercle vert généré.
Voir vidéo : `cedric.cnam.fr/~porumbed/exo7.mp4`

Exercice 8 **(bonus)** Modifier le programme précédent pour ajouter la fonctionnalité suivante. Après avoir dessinné 15 cercles verts, la méthode `draw()` arrête de générer de nouveaux cercle et commence à augmenter le rayon des 15 cercles verts existants, d'une unité à chaque appel `draw()`. Lorsque l'un des 15 cercles vert touche un cercle rouge, on commence à diminue le rayon des cercles verts, d'une unité à chaque appel `draw()`. Le programme s'arrête complètement lorsque le rayon atteint la valeur 0.