ISN S5 : Initiation à la programmation : Processing 2

But :
Etre capable de mettre en œuvre des variables de type booléen, entier et flottants : savoir les créer, les initialiser, les modifier. Créer un programme Processing comprenant une animation simple. Savoir mettre en œuvre une structure condition.

Dans ce TP, nous allons voir comment créer une animation. Nous aurons pour cela besoin d'utiliser des variables…

1. Animation de base

I) Commençons avec un programme court, créant une animation. Créez le programme ci-contre sous Processing, enregistrez-le sous le nom « PrgDeBase » et exécutez-le. Expliquez ce que fait ce programme :

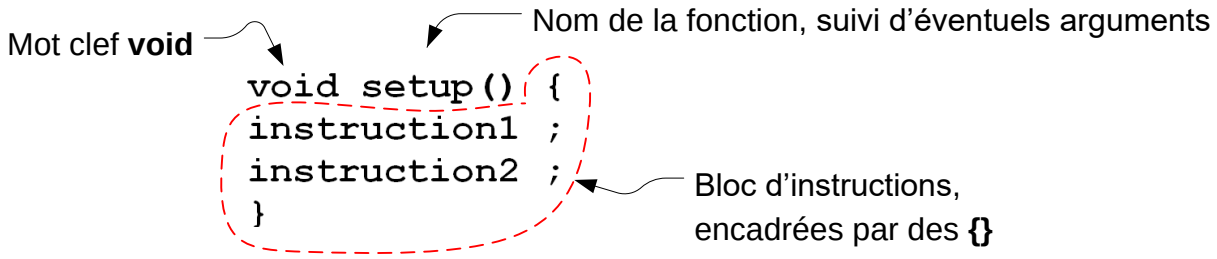
```
void setup() {
                size(500, 300);
              }

void draw() {
  if (mousePressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX,mouseY,80,80);
}
```

Il y a beaucoup d'éléments nouveaux à découvrir dans ce programme, mais il vous permet de voir qu'il est très facile de créer une animation sous Processing (c'est pour cela qu'il a été créé).

On note dans ce programme des instructions déjà étudiées : size, pour fixer la taille du dessin ; fill, pour choisir la couleur de remplissage ; ellipse, pour tracer un cercle.

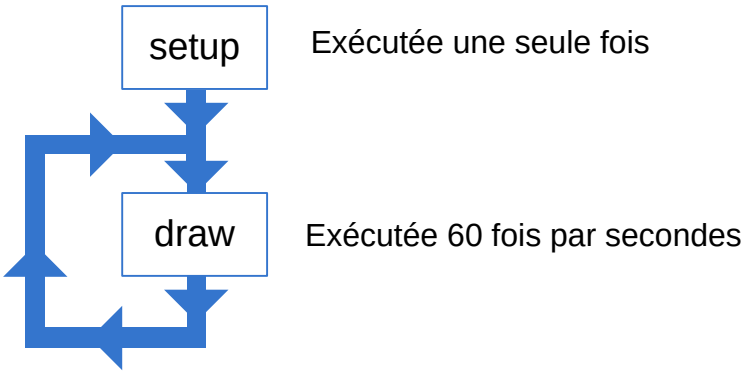
On constate : « void setup() {} et void draw() {} ». Il s'agit de fonctions, mais nous allons passer rapidement sur leur syntaxe, nous y reviendrons dans un autre TP. Aujourd'hui, nous allons admettre qu'elles s'écrivent ainsi :



Les 2 fonctions utilisées permettent :

1. d'initialiser le programme une seule fois avec setup
2. de répéter indéfiniment la séquence de dessin avec draw.

Par défaut, la répétition de draw et de 60 fois/sec :



On peut changer cette valeur avec l'instruction frameRate=taux de rafraichissement, ou connaître le nombre de rafraichissement effectués avec frameCount.

Remarque : l'utilisation de setup et **draw** est un choix fait par l'équipe de développement de Processing pour faciliter la réalisation d'une animation. Il faut bien comprendre que lorsque le programme est traduit en langage Java, il y a un algorithme (masqué pour nous) qui indique qu'il faut au démarrage du programme appeler la fonction setup une fois, puis répéter l'appel de la fonction **draw** au rythme choisi, jusqu'à l'appui sur la touche escape.

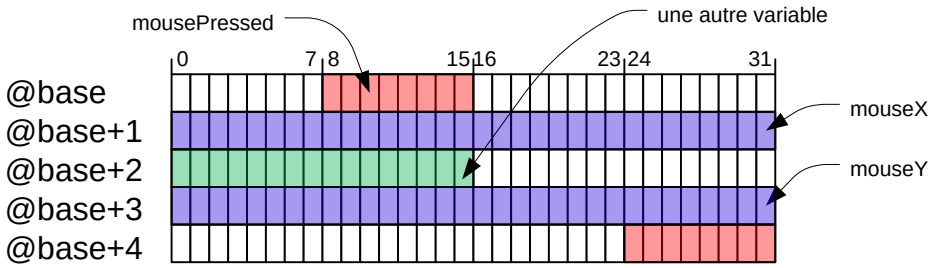
On note l'utilisation d'éléments **en rose**, les variables. Comme son nom l'indique, la valeur d'une variable peut changer au cours du temps. Pour afficher une variable, il suffit d'utiliser la fonction **println**(nomDeMaVariable) ; Par exemple **println**(« la variable mouseX vaut : » + **mouseX**) ; affichera en console « la variable mouseX vaut : 123 ».

Rq : l'opération « + » va réaliser ici la conversion de la valeur de la variable en un texte, puis l'ajouter au texte précédent.

II) En utilisant println(…) ; dans votre programme, indiquez la gamme de valeur possible pour les variables :
- mousePressed
- mouseX :
- mouseY :

Conclusion : une variable possède un **type**, elles ne sont pas toutes identiques. Selon le type de la variable, elle sera codée différemment, et utilisera plus ou moins de place en mémoire.

Le nom de la variable permet de pointer vers un espace mémoire ou se trouve physiquement la valeur de votre variable. Par exemple, voici un extrait possible de la mémoire de votre ordinateur :



III) Indiquez le nombre de bits utilisé pour stocker :
- un booléen :
- un entier :

Remarque : La communication entre le processeur et la mémoire se faisant par paquets de 32 bits (ou 64 bits selon votre processeur), on ne veut pas fragmenter la mémoire en mettant des variables à cheval sur 2 lignes.

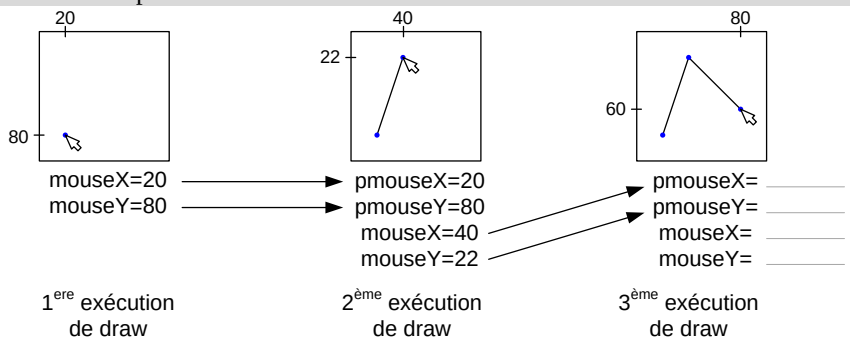
2. Utilisons les variables

Processing propose des variables toutes faites bien pratique comme **mouseX** ou **mouseY**, mais il en existe bien d'autres, par exemple **width**, **height**, **pmouseX**, **pmouseY**, **keyPressed, key**, **keycode**…

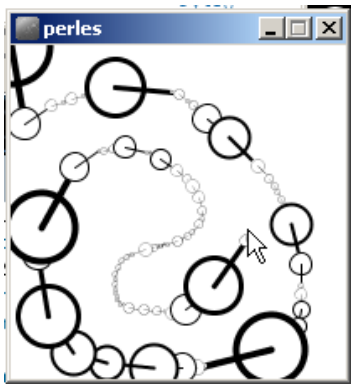
IV) En utilisant l'aide, indiquez le type et le rôle des variables :
- width:
- height
- keyPressed

Nous allons maintenant utiliser en plus des variables mouseX et mouseY leurs valeurs précédentes contenues dans pmouseX et pmouseY pour faire un programme simple de tracé, en utilisant l'instruction **line();**

V) Complétez la dernière étape du tracé désiré :



VI) Proposez un programme pour réaliser l'image ci-contre: →



Voici une légère modification du programme, qui tient compte de la vitesse de déplacement de la souris, et ajoute un cercle.

VII) Comment pourrais-t-on calculer simplement une estimation de la vitesse de la souris ? Proposez une formule mathématique :

3. Créons nos variables

Nous pouvons également créer nos variables. En Java, comme dans tous les langages, ils existent des types indispensables appelés **primitif**. A partir de ces types, on peut construire des variables plus sophistiquées par la suite.

Il y en a 4 principaux : vous connaissez les booléens (**boolean** sur 8 bits), les nombres entiers (**int** sur 32 bits), les caractères (**char** sur 16 bits) pour stocker une lettre et les nombres à virgules (**float** sur 32 bits) pour stocker les réels.

Pour créer une variable, il faut obligatoirement la **déclarer** en indiquant son **type** puis son **nom**, et on peut l'**initialiser** (lui donner une valeur par défaut), même avec un calcul. La déclaration permettra de réserver la place nécessaire dans la mémoire pour votre variable.

boolean obstaclePresent; // on utilisera une variable booléenne nommée obstaclePresent
obstaclePresent = true; // initialisation de la variable

On peut aussi tout faire en une seule fois :
boolean lancerAnimation=false ; // déclaration et initialisation

On ne doit pas mettre d'espace ni d'accents dans le nom de la variable, ni utiliser de mots clefs réservé du langage. Donnez un nom clair à vos variables, comme « couleurRemplissage » proscrivez « toto ». Par convention, il faut commencer son nom par une minuscule et pour un nom composé de plusieurs mots, mettre une majuscule au début de chaque mot, par exemple « ageDuCapitaine ».

Quelques exemples de déclarations :
int compteur = 3;
char caractereClavier = 'A';
float deuxPi = 6.28319;

Java refuse de convertir un type vers un autre s'il y a perte d'information. Dans ce cas, on peut forcer une conversion de variable en précisant entre parenthèse le type de variable désiré. Par exemple :
int taille=0;
taille=100*sqrt(2); // donne une erreur « can not convert from float to int » car la fonction racine (square root en anglais) donne un nombre à virgule, donc un flottant.
Par contre **taille=(int)(100*sqrt(2));** // fonctionne, taille vaudra 141

Les calculs avec les variables ne doivent pas vous faire croire que vous ne comprenez plus rien aux maths… Ce ne sont pas des équations mathématiques mais essentiellement des modifications de mémoire. Par exemple :
int x=3;
positionX=positionX+5;
Ne se traduit pas par positionX-positionX=5 ⇔ 0=5 !!!
Cela indique que la nouvelle valeur de positionX va valoir l'ancienne valeur+5 (donc 8 ici). Autrement écrit, positionX(n+1)=positionX(n)+5, où n est l'état actuel de la machine, n+1 son état futur après une étape.

VIII) Indiquez les valeurs de la variable x à chaque ligne de code

- int x=2; // x=
- int y=4; // x=
- int x=y; // x=
- int y=8; // x=
- int x=x+3; // x=

4. Animons une balle

Vous allez maintenant créer un programme qui déplace une balle horizontalement sur l'écran.
Cahier des charges :

- La balle se situe au milieu de l'écran.
- sa position en x est repérée par la variable positionX
- l'incrément du déplacement est contenu dans la variable deltaX

IX) Complétez le programme pour réaliser ce cahier des charges

```
// déclaration des variables

void setup() {
        size(500, 300);
        }
void draw() {

        }
```

La balle va rapidement sortir de l'écran une fois le programme lancé. On va donc la faire rebondir sur les bords. Pour cela, il va falloir une structure condition. Il y en a une dans le premier programme étudié, mais nous l'avons passé sous silence… il est temps de changer cela !

Voici la **structure condition**, qui est celle du langage Java, en pseudo langage et en Java :

Si la **condition** est vraie,
alors faire
    instructions1
    instructions2
sinon faire
    instructions3
    instructions4

```
if (condition) {
        instructions1 ;
        instructions2 ;
}
else    {
        instructions3 ;
        instructions4 ;
}
```

Observez la séparation des blocs d'instructions par les accolades {}, que des points virgules terminent les instructions, mais qu'il n'y a pas de points virgules pour terminer la structure if.

Ctrl-t permet d'obtenir un formatage automatique du code source… utilisez-le !

La condition doit donner un **booléen**. C'est souvent le résultat d'une comparaison, par exemple (mouseX>50).
Vous pouvez comparer avec les opérateurs : **!= (différent),<,<=,==(égaux),>,>=**
Attention, le test d'égalité contient 2 signe égal : (chat==chien) est correct, (chat=chien) est une erreur !

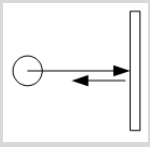
Si on travaille avec un booléen directement, il est inutile de tester son état : if (ilPleut==true) est équivalent à if (ilPleut). De même, on n'écrit pas if (ilPleut=false), mais plutôt if (!ilPleut). **Le point d'exclamation représente l'inverseur booléen**. Vous pouvez combiner les booléens avec **&&** (ET logique), || (OU logique, s'obtient par alt gr -6)
Par exemple, if (ilPleut||ilVente) {mettreImperméable=true ;}

X) Complètez le programme pour réaliser le cahier des charges avec rebonds sur les murs

```
// déclaration des variables


void setup() {
    size(500, 300);
        }
void draw() {

        }
```



XI) Modifiez le programme pour que la balle se déplace en x et en y et qu'elle rebondisse sur tous les murs.

XII) Remplacez la balle par votre petit extra terrestre. Changez la couleur de ses yeux en fonction de sa position sur l'écran, bref amusez vous, il faut que cela bouge !