////////////////////////

# DESIGNING SYSTEMS

//////////////////////////////////////////////////////////////////////////

# BASES : INTERACTION_01

//////////////////////////////

## Sommaire

- Dessiner avec la souris
- Utiliser les touches de clavier
- Comment créer des motifs interactifs.

## Concepts / Mots Clés

interaction souris, clavier, map() & constrain()

## Dessiner avec la souris

Les coordonnées de la souris dans l'espace de dessin peuvent être récupérer avec les variables propres à Processing :

mouseX - Position de la souris sur l'axe X

mouseY - Position de la souris sur l'axe Y

Notez que **mouseX** & **mouseY** sont des variables propres à Processing. C'est-à-dire qu'ils stockent les coordonnées de la souris et nous pouvons donc se servir de ces valeurs tout simplement pour dessiner un objet graphique.

Voir le sketch *Mouse_Draw*

```
ellipse(mouseX, mouseY, 10, 10); // On dessine l'ellipse là où se trouve la position de notre souris dans l'espace de dessin.
```

Nous allons maintenant améliorer notre petit système de dessin. Vous avez sans doute remarquer que nous dessinons l'ellipse à chaque fois qu'on bouge la souris. Il serait plus pratique d'avoir un système où on dessine uniquement lorsqu'on appuie sur la souris. Pour ce faire on va se servir d'une structure conditionnelle qui permettra de vérifier si nous appuyons sur le button de notre souris.

Voir le sketch suivant, *mousePressed_01*

```
if (mousePressed == true) { // Structure conditionnelle
...
ellipse(mouseX, mouseY, 10, 10);
}
```

La structure conditionnelle teste une condition - ici, il s'agit de si l'on appuie sur la souris. Si oui, le programme exécute le code qui se trouve entre les deux accolades. Sinon, il ne fait rien. Simple !

Processing inclut deux autres variables ; pmouseX & pmouseY. Ces variables contiennent les coordonnées précédentes de la souris. Nous pouvons donc se servir des ces variables pour dessiner des lignes car nous disposons des 4 points nécessaires.

Voir le sketch *Mouse_Draw_Line*

```
line(mouseX, mouseY, pmouseX, pmouseY);
```

## Utiliser les touches de clavier

Les touches de clavier peuvent s'avérer très utiles pour intervenir sur un programme. Un exemple simple, c'est la possibilité de sauvegarder une image en appuyant sur une touche désignée par nous même. Mais nous pourrions imaginer bien d'autres usages comme par exemple changer la taille d'une forme, sa couleur ou effacer complètement un dessin pour recommencer à nouveau. Alors, il existe deux manières principales pour se servir des touches et vous allez voir qu'ils n'ont pas de tout le même comportement. Premièrement, regardons la variable keyPressed qui fonctionne comme mousePressed.

Voir le sketch *KeyPressed_01*

```
  if (keyPressed == true) {

    fill(255, 0, 0);

    rect(width/2, height/2, 200, 200);

  }
```

Vous remarquez qu'à chaque fois qu'on appuie sur n'importe quel touche du clavier, on fait apparaître le rectangle. Ensuite, lorsqu'on lâche la touche, il disparaît.

Maintenant, dans le même sketch, vous verrez un bloc de code en bas dans la partie FUNCTIONS. Activez ce bloc de code en supprimant /* .....*/ et désactiver dans le bloc de code draw() la structure conditionnelle. Vous verrez que lorsqu'on appuie sur une touche le rectangle s'affiche rapidement. Nous n'avons donc pas le même comportement. En fait, dans la partie FUNCTIONS, nous avons ce qu'on appelle une fonction d'événements qui s'exécute qu'une fois dans le programme après avoir lancer l'événement. Ici, il s'agit d'afficher un rectangle lorsqu'on appuie sur une touche. Processing donc affiche cet événement en permanence (tant que nous restons appuyer).

Il est possible de désigner n'importe quelle touche de clavier pour lancer un événement. Voir les sketchs *keyPressed_02 - keyPressed_05* pour voir les différentes possibilités.

Le sketch *keyPressed_05* est un système de dessin plus avancé que notre premier sketch, *Mouse_Draw*. Grâce aux touches de clavier, nous pouvons changer la couleur, la taille de la forme et aussi faire une sauvegarde de l'image. Essayez de rajouter d'autres fonctionnalités à ce système : d'autres couleurs, d'autres formes. Vous pouvez essayer aussi de créer une forme plus complexe en écrivant une fonction dédiée.

## Comment créer des motifs intéractifs

Voir le sketch *Grid_01*. Dans ce sketch on se sert des variables **mouseX** et **mouseY** pour interagir avec une grille de cercles. La grille est dessinée à l'aide d'une boucle imbriquée (les FOR loops). Nous avons déjà regardé ensemble des sketches de ce type - revoir les sketches qui se trouvent dans le dossier SUPERFORMS.

```
for (int i = 0; i<=width; i+=50) {
 for (int j = 0; j<=height; j+=50) {
      ellipse(x, y, mouseX, mouseY); // USE THIS LINE FIRST.

      // Then check these. What is the difference?
      //float mx = constrain(mouseX, 50, 100);
      //ellipse(x,y,mx,mx);
      //float mx = map(mouseX,0,width,50,100);
      //ellipse(x,y,mx,mx);
    }
  }
```

Lorsqu'on bouge la souris dans l'espace de dessin, les cercles changent de taille et de forme. On va développer ce premier système pour explorer les possibilités de création de motifs. Tout d'abord on va regarder deux fonctions très utiles. Dans le sketch, vous notez trois lignes de codes qui sont en commentaires, (donc qui ne sont pas prise en compte dans l'exécution du code). Activez chaque ligne pour voir ce qu'il se passe. Essayez de comprendre leur fonctionnement. (N'oubliez pas de désactiver les autres lignes)

## map() & constrain()

`constrain()` nous permet de limiter une variable (mouseX dans l'exemple) avec une valeur minimum et maximum, (min = 50 / max = 100).

map() nous permet d'appliquer à une variable une plage de valeurs. Notez que nous avons 5 arguments entre parenthèse : le premier désigne la variable à laquelle on applique ce changement (mouseX), les 2 suivants désignent la plage de valeurs initiales de la variable - donc 0 à width (largeur de notre fenêtre). Finalement les deux derniers arguments désigne la nouvelle plage de valeurs que nous souhaitons appliquer (50 à 100). Maintenant, vous constatez que lorsqu'on bouge la souris sur l'axe X, nous augmentons la taille de nos cercles de 50 pixels jusqu'au 100 pixels. C'est-à-dire que lorsque la souris se trouve au bord de la fenêtre à gauche, la position x = 0 et la taille des cercles est égal à 50. Lorsque la souris se trouve au bord à l'extrême droite, la position x = width et la taille des cercles est égal à 100. La fonction map est beaucoup utilisée et s'avère très pratique pour convertir une plage de valeurs à une autre plage. Jouez avec les valeurs des arguments pour bien comprendre son fonctionnement.

Dans le sketch *Grid_02* nous allons créer notre grille de formes d'une manière légèrement différent. Regardez bien les boucles for. Vous remarquez une variable qui est nommé gridRes. Cette variable nous l'avons déclaré en haut dans la partie GLOBALS et avec une valeur de 10. Cette valeur détermine la résolution de notre grille (le nombre de formes dessinées). Retour à nos boucles for. Vous remarquez également la création de deux variables, xPos & yPos qui vont déterminer la position de nos formes. Le calcul se fait par rapport à la valeur gridRes (10) et la largeur et hauteur de notre fenêtre. Vous pouvez facilement faire le calcul vous même pour mieux comprendre. Par exemple, lorsque x = 1 dans la boucle FOR, xPos = width(600)/10 * 1 qui fait 60. Lorsque x = 2 (souvenez-vous que la valeur de x augment grâce à notre boucle) xPos = width/10 * 2 qui fait 120 et ainsi de suite. Pourquoi utilisez cette méthode ? Tout simplement parce que  le concept de grille est plus explicite et que nous n'avons qu'une variable à changer (gridRes) si on veut modifier la grille.

Nous allons maintenant faire évoluer ce système pour explorer d'autres pistes graphiques. Voir le sketch *Interactive_Squares*. Le système est le même sauf que nous avons ajouté un comportement différent. À l'aide de la fonction map() nous faisons pivoter le rectangle un tour complet lorsqu'on bouge la souris de haut en bas sur l'axe Y. Un petit problème pour vous ! Est-ce que vous pouvez essayer de rajouter un deuxième rectangle qui tourne en même temps que l'autre mais dans l'autre sens ? En plus, faire diminuer sa taille alors que l'autre grandit !