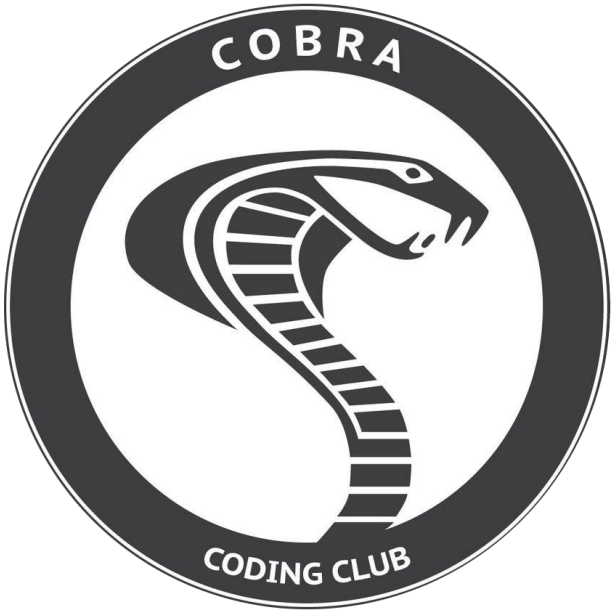
Coding Club - Sugarscreen

Version 1.2



DIVERSITY by EPITECH

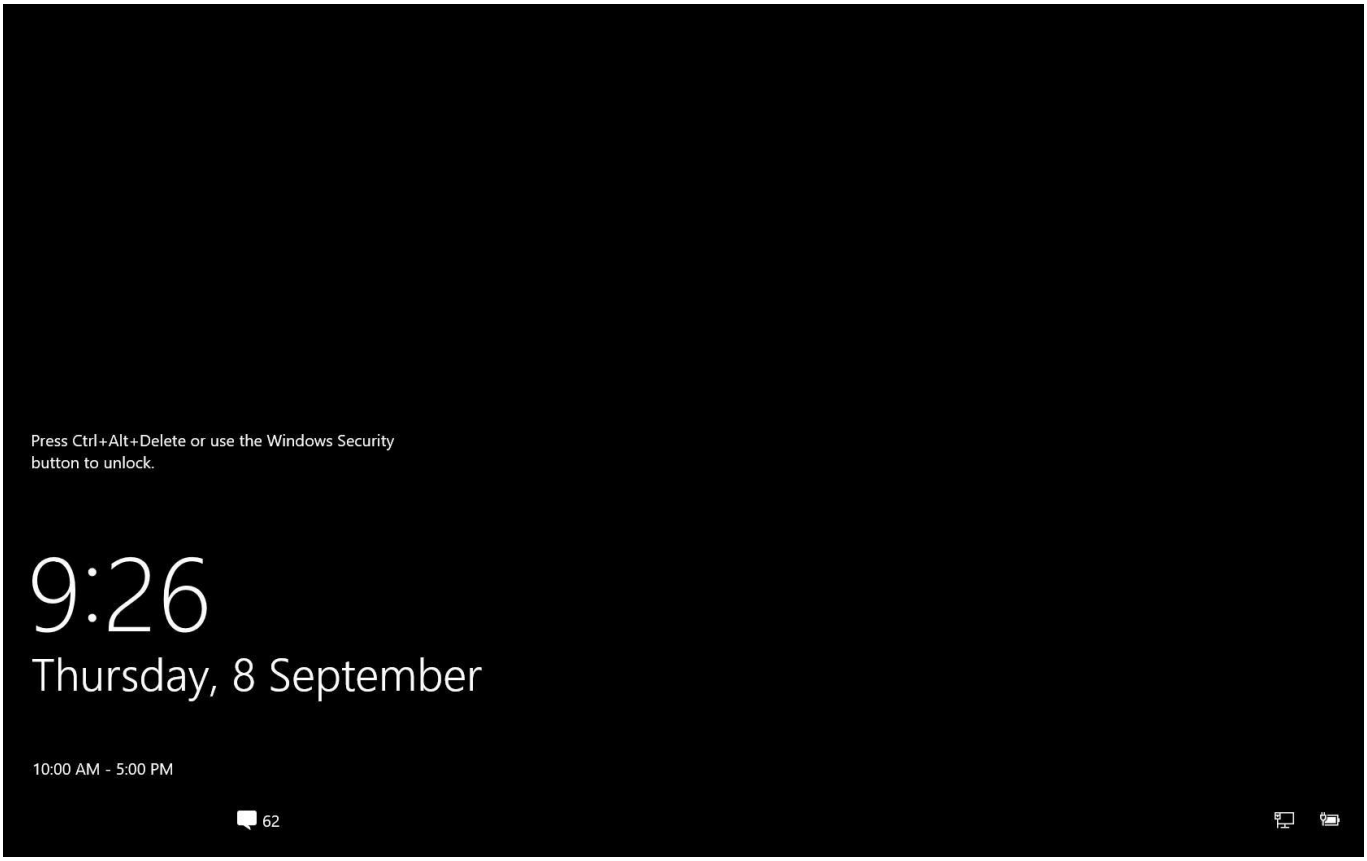
Introduction

Processing est un outil permettant d'apprendre à coder par la création graphique.
Il est utilisé par des étudiants, artistes, chercheurs et curieux pour apprendre ou créer rapidement des visuels.

Par exemple, la chaîne YouTube Coding Train, qui fait de la découverte de l'algorithmie, se sert souvent de Processing dans ses vidéos.
Dans une de ses vidéos, il crée un Star Field (affichage d'étoiles) !

Aujourd'hui, vous allez apprendre à coder avec Processing.

Markus Sugarberg s'est fait hacké son jeu récemment et veut renforcer la protection des ordinateurs de sa boite. Il a créé un système super sucrisé pour les protéger, mais les employés se plaignent de l'aspect de l'écran. Aidez-les à égayer leurs journées en créant un super écran de veille.



Consignes

- Vous pouvez utiliser la documentation de Processing pour en apprendre plus : Processing
- Si vous avez des questions ou des problèmes trop salés essayez de les résoudre entre vous avec de l'eau, puis, posez des questions au Cobra.

Les bases du glucose

Pour commencer cette mission, on va essayer de reproduire ce fond d'écran très connu avec un objet qui rebondit contre les bords de l'écran. La première étape, c'est d'afficher quelque chose et de le faire bouger.

Une mise en place sucrée

C'est l'une des rares notions compliquées de Processing. setup() et draw() sont des fonctions particulières que vous devez écrire pour avoir du mouvement à l'écran.

Commencez par écrire simplement :

```
void setup()
{
  size(600, 600); // définit la taille de la fenêtre
  background(100); // définit la couleur du fond de la fenêtre, ici 100 veut dire
gris
  fill(255, 0, 0); // remplit le cercle en rouge
  ellipse(width/2, height/2, 50, 50);

}

void draw()
{
  fill(0, 0, 255); // remplit le cercle en bleu
  ellipse(width/2, height/2, 50, 50);
}
```

On constate que le cercle au centre est bleu. C'est parce que la fonction draw() est appelée **après** setup().

> Comme son nom l'indique setup() sert à préparer le reste du programme. Il passera donc une seule fois par cette fonction au tout début.
>
> A l'inverse draw() (se traduit par dessin) est appelé plusieurs fois pour réafficher à chaque fois ce qu'on lui demande (elle est appelée 60 fois par seconde !)

Ajoutons notre grain de sucre... Essayons de faire bouger quelque chose maintenant !

Créez une `variable` qui sera un entier et augmentera de 1 à chaque fois que l'on passe par draw(), puis utilisez le comme valeur `y` dans votre fonction ellipse().

Berk n'est-ce pas ? Si vous avez réussi vous avez une sorte de rond qui dégouline du haut de la fenêtre.

Quel manque de gout, corrigez ce problème !

> Vous savez ce que fait `background` ?

Lancé de sucre

Bon on va commencer à faire des choses plus douces mais il nous faut les bons outils. Désormais, pour stocker des coordonnées, nous allons utiliser `PVector`.

Premièrement remplaçons notre `variable` par `PVector` pos :

```
PVector pos = new PVector(0, 0); // bravo vous avez découvert une nouvelle
dimension !
```

> *C'est quoi ça "new" ?* De la magie salée !
>
> Plus sérieusement, c'est un mot magique qui permet de créer un nouveau PVector. Je vous expliquerais ce que cela implique plus loin, on y arrive 😃

`Pos` contient maintenant un `x` et un `y` qui valent 0. Pour y accéder il faut écrire `pos.x` et `pos.y`.
Modifiez votre code pour le faire fonctionner avec pos.

Bon on a un bonbon rond qui bouge ! C'est pas mal !
Dites-moi vous l'avez laissé bouger assez longtemps pour voir ce qu'il se passait ? Comment fait-on pour qu'il ne sorte pas de l'écran ?

Il va vous falloir deux choses :

1. A quel moment rebondir ?
2. Comment rebondir ?

La première partie est une **condition**, on va chercher à faire une action, seulement si la **condition** est remplie.

En l'occurrence la condition se résume ainsi : *Si la balle passe le bord de la fenêtre*.

La deuxième partie est une action comme toutes celles que l'on a fait jusque-là. Il va falloir inverser le sens de déplacement du cercle.

Faites rebondir le cercle !

> conseil : si j'augmente de 1 le cercle pour le faire aller vers le bas de combien dois-je l'augmenter pour qu'il remonte ?

Bon, maintenant, un exercice un peu salé pour finir cette partie 😃**Faites en**

sorte que :

1. **votre cercle se déplace en x *et* en y**
2. **votre cercle rebondisse sur toutes les parois de la fenêtre**
3. **votre cercle commence avec des vitesses aléatoires en x et en y**

> L'aléatoire c'est une base en programmation il y a sans doute quelque chose pour ça...

> Cet exercice est volontairement compliqué pour vous forcer à réfléchir !

Les sucres en confinement

Jusqu'ici on a utilisé une méthode appelée "itératif". On va maintenant s'intéresser aux classes et la "programmation orientée objet".

Les moules à bonbon

Nous allons donc commencer par créer une classe.

Une `classe` c'est ce qui nous permet de créer des objets. Imaginez la `classe` comme le moule avec lequel on fait un bonbon.
On écrit une classe comme ceci :

```
// on crée une Classe qui s'appelle "Ball"
class Ball
{
  // Les objets créés avec cette classe auront toutes ces "variables membres"
  PVector pos;
  int size;

  // les classes ont une fonction spéciale appelée "Constructeur" qui donne toutes les valeurs de bases aux variables un peu comme le setup
  Ball(int _size) // A la création on demandera la taille de la balle (le "_" permet de différencier la variable membre size du paramètre _size)
  {
    pos = new PVector(width/2, height/2);
    size = _size;
  }

  // display est une fonction membre
  // display doit afficher un cercle aux coordonnées pos de taille size par size
  void display()
  {
    ...
  }
}
```

Remplissez la fonction display() pour que la balle puisse être affichée 😃

Faire un bonbon c'était drôle mais j'en fais quoi ?

Maintenant, on a notre moule ! Il nous faut donc créer le bonbon **avec** le moule. Vous vous souvenez de la formule de magie salée ?

```
// Je crée un objet (variable) appelé "ball" qui est une nouvelle Ball de taille 50
Ball ball = new Ball(50);
```

Eh oui ! "`new`" sert à **construire** un objet à partir de son plan.
Maintenant l'objet peut modifier ses propres variables et se débrouiller tout seul. Il a juste besoin qu'on lui dise quand le faire. On peut donc appeler ses fonctions membres comme suit :

```
// j'appel la fonction "display" de l'objet "ball" créé plus tôt
ball.display();
```

Allez ! On passe aux choses sérieuses ! eheh~

1. **réécrire les fonctions setup() et draw() pour afficher votre cercle**
2. **écrire une fonction membre à Ball qui la fait bouger selon une nouvelle variable spd**
3. **écrire une fonction stayIn() qui empêche la balle de sortir de la fenêtre**

C'est le feu et si j'en faisait 2... ou 5000

Maintenant que vous avez de quoi construire un bonbon, vous pouvez en construire à peu près autant que vous voulez...

Hey ! Oh là ! Revenez vite ! Je vous vois avec vos :

```
Ball ball1 = new Ball(50);
Ball ball2 = new Ball(50);
Ball ball3 = new Ball(50);
Ball ball4 = new Ball(50);
Ball ball5 = new Ball(50);
```

On dirait le code d'une salière !

Ce n'est pas avec ça que vous allez en construire des centaines, voire des **milliers** ! Il existe en Processing quelque chose de cool qui s'appelle les tableaux et ça donne ça :

```
Ball balls[] = new Balls[100] // crée 100 Ball

// dans setup :
  for (int i = 0; i < balls.length; ++i) // i ira de 0 au nombre de balls (100)
  {
    balls[i] = new Ball(10); // crée une nouvelle Ball dans la case i du tableau
balls
  }

// dans draw :
  for (int i = 0; i < balls.length; ++i)
  {
    balls[i].display(); // afficher la ball de la case i du tableau balls
    ...
  }
```

> Ici "++i" remplace "i = i + 1" et veut dire exactement la même chose. On appelle ça un "sucre syntaxique" (miam)

Appelez les autres fonctions de Ball à la place des "..."

Et voilà pour les classes ! Vous avez maintenant quelque chose qui ressemble à un fond d'écran de veille Windows d'une autre époque !

Vers le diabète et au-delà

Voilà, notre Sugarscreen est fonctionnel, maintenant c'est à vous de rajouter la cerise sur le gâteau !

- Images de bonbons à la place des boules.
- Image de fond (chocolaterie par exemple).
- Mouvement aléatoire des bonbons.
- Rendre les bonbons cliquables pour les collecter, incrémentation de son score, changement de couleur du bonbon etc... SOYEZ CREATIF !
- Ajouter des bonbons en appuyant sur une touche.

Merci pour votre Sugarscreen, attention de bien se laver les dents pour éviter les caries ! 😄