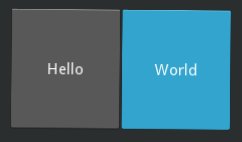
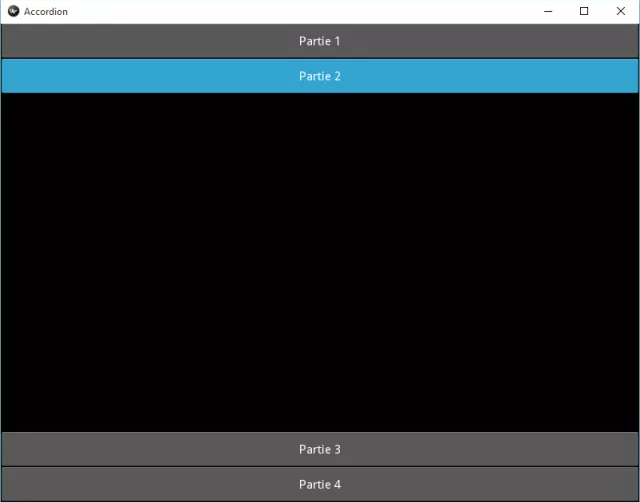
**Synthèse personnelle**

Après avoir bien défini en quoi consistait notre projet, nous nous sommes rapidement divisés les tâches. Léo devait initialement s’occuper de l’interface graphique de notre application. Quentin et moi devions nous occuper du code de ces applications. Mais après mure réflexion nous avons compris que les codes à écrire ne seraient pas très long, nous avons donc préférer laisser ce travail à une seule personne. Il s’est aussi avéré que Léo ne pouvait pas télécharger Kivy sur son mac. Nous avons donc décidé de laisser Léo au code des applications, et de nous concentrer avec Quentin sur l’interface graphique.

**Le travail réalisé avec Quentin**

Pour réaliser nos différentes applications en Python nous avions décidé d’utiliser la bibliothèque Kivy. Pour commencer nous avons réalisés de nombreuses recherches et de nombreux tests.  
  
J’ai pour ma part beaucoup apprécié le site <http://www.mathslallemand.fr> qui propose des tutoriels en français sur Kivy. Ces tutoriels sont simples même pour des débutants.  
A l’aide de ces tutoriels j’ai pu tester plusieurs fonctionnalités de Kivy :  
- La création d’une interface graphique simple : - La création de plusieurs boutons :



- L’utilisation de l’accordion :

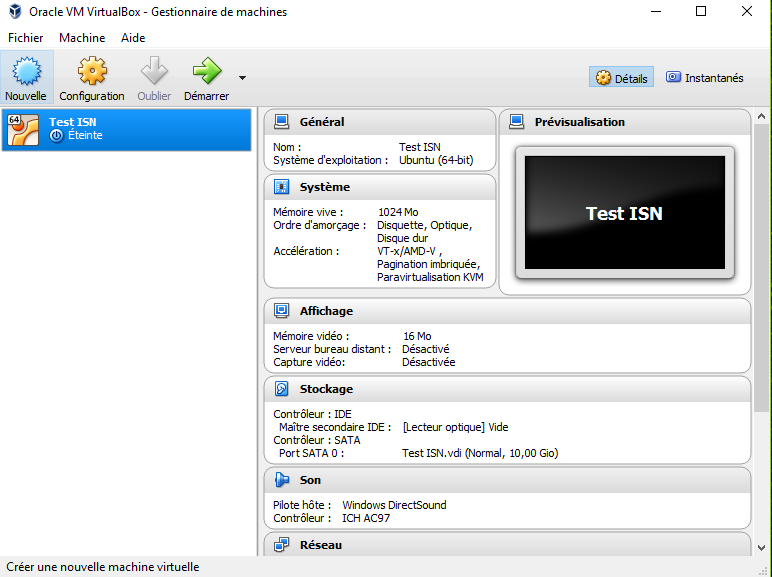
**Mon environnement de travail**

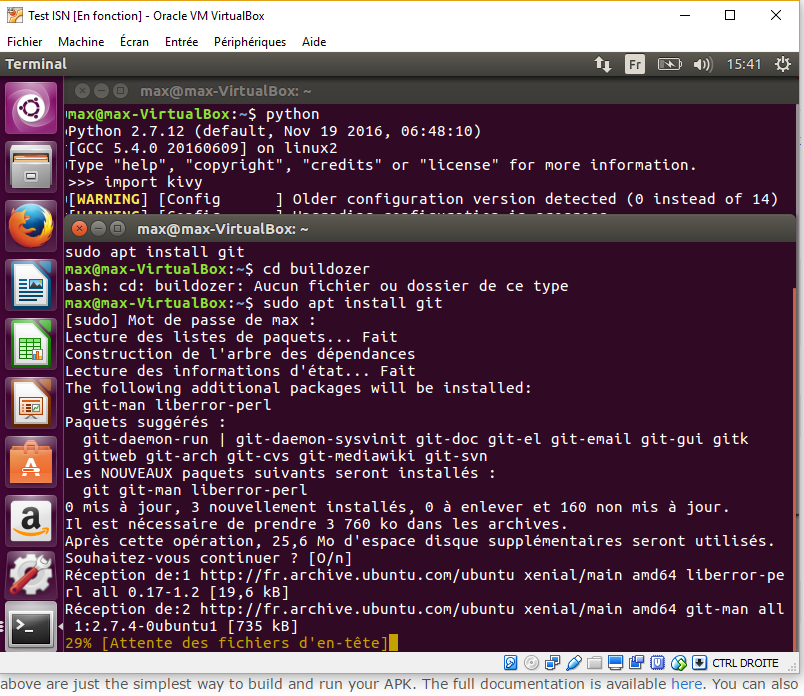
Le but ici, est de décrire quels outils j’ai pu utiliser ainsi que la méthode de travail que j’ai employée.

**PyCharm**  
Tout d’abord mon environnement de travail était composé de PyCharm, un environnement de développement intégré qui permet de programmer en Python. De plus PyCharm offre un débogueur très pratique.

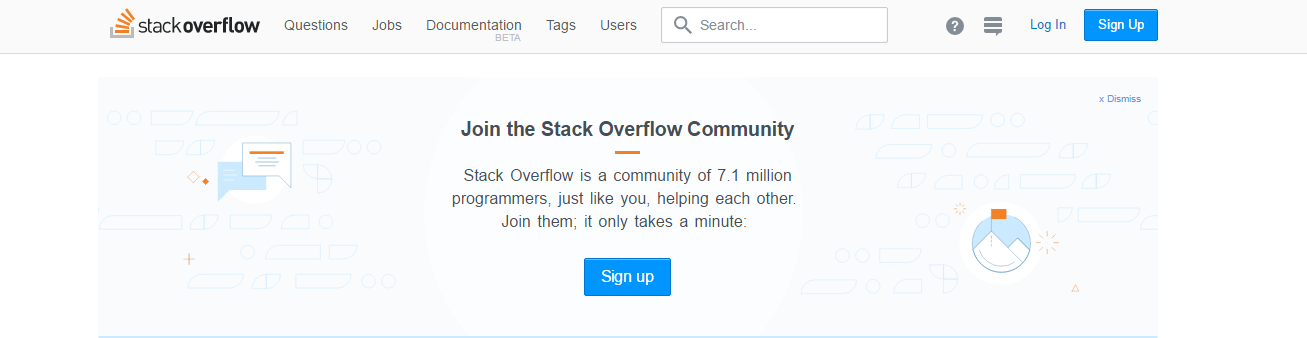
**Kivy**  
J’ai ensuite utilisé Kivy pour pouvoir réaliser notre application sur ordinateur pour pouvoir, par la suite, migrer sur smartphone. Kivy est une bibliothèque open source pour Python utile, pour créer des applications sur smartphone. Kivy est le principal framework développé par la société Kivy. Un framework est une infrastructure de développement.



**Machine Virutelle : OracleVM Virtual Box**Le but de notre projet étant de développer notre application sur mobile, nous nous sommes renseignés sur la procédure à suivre pour pouvoir transférer notre application d’un ordinateur à un smartphone. Nos recherches ont mis en évidence que nous devions utiliser un module de Kivy, **KivyBuildozer**, disponible uniquement sous Linux. J’ai donc installé une machine virtuelle avec Linux dessus.

J’ai ensuite installé en apk (depuis le terminal de linux) les fichiers et les commandes manquants.

**Stack Overflow**Stack Overflow est un site web proposant des questions et réponses sur un large choix de thèmes concernant la programmation informatique. Ce site nous a beaucoup aidé, car de nombreuses personnes ont commis les mêmes erreurs que nous, et ont finalement trouvé une solution à leur problème.

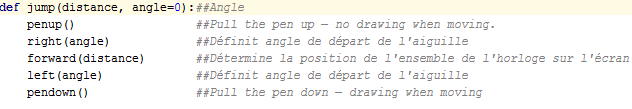


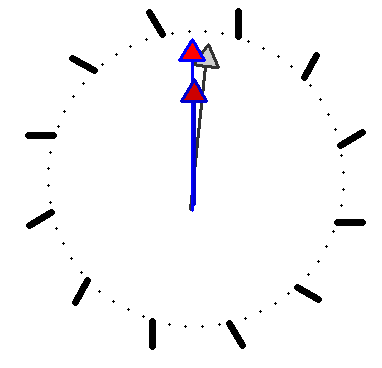
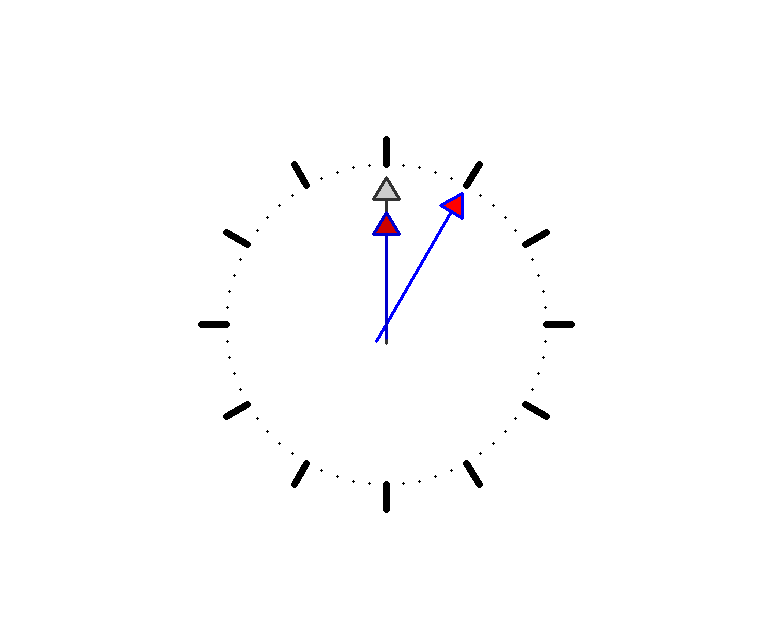
**Communication au sein du groupe**Pour partager nos opinions, nos travaux ainsi que toutes nos idées, nous avons utilisés plusieurs applications propices au partage d’informations comme : GitHub, Skype, Messenger.



**Ma Méthode de Travail**Tout d’abord je me suis penché sur le « corps » de notre 1er programme : le chronomètre. Avec Léo nous avons réalisé très rapidement un code faisant tourner notre chronomètre. Ensuite avec Quentin nous nous sommes penchés sur l’interface graphique, qu’il fallait mettre en relation avec le code. Nous sommes pour cela penchés sur le langage Kivy.  
Pour avoir une application esthétiquement jolie, nous avons eu pour idée d’avoir pour interface graphique une horloge avec des aiguilles tournant au fil du temps. Pour cela nous avons d’abord codé le design de l’horloge pour ensuite coder les « objets » que sont les aiguilles.  
Il a fallu ensuite relier l’interface graphique au code, ce qui nous as menés à de nombreuses difficultés.

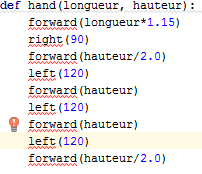
**Les fonctions utilisées**



La fonction jump définit l’image d’arrière-plan, l’horloge. Elle prend comme paramètres la distance du milieu du cercle ainsi que l’angle d’inclinaison des petits bâtonnets.  
Quand l’angle est définit à 0 degrés : Quand l’angle est définit à 15 degrés :  


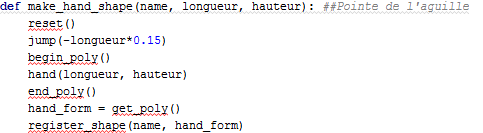
Les fonctions penup() et pendown() permettent de dessiner graphiquement le mouvement des aiguilles ou non (comme un stylo poser sur le papier ou levé).  
forward() détermine la position de l’horloge sur l’écran. Ici la position de l’horloge est centrée au centre de l’écran.  
Left() et right() déterminent les positions de départ des aiguilles (leur angle de départ) ainsi que la position relative du cercle au niveau de l’interface graphique de l’utilisateur.

Hand : Définit l’aiguille, sa hauteur, son épaisseur, les angles entre son corps et sa tête.



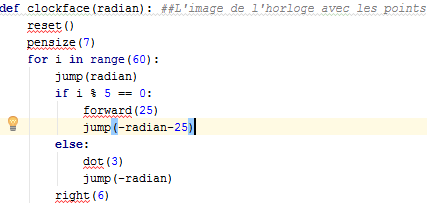
Forward() bouge l’objet dans une direction spécifique, avec un angle spécifique.  
Left() permet de manipuler la tête de l’aiguille par rapport à son corps, de pouvoir l’incliné vers la gauche.  
Right() est similaire à left(), mais vers la droite cette fois-ci.

Make\_hand\_shape : Définit la pointe de l’aiguille, l’aiguille et son mouvement.



begin\_poly() commence à sauvegarder la position des sommets d’un polygone, end\_poly() stoppe, et get\_poly () permet de réutiliser cette information.  
jump() permet de ne pas passer par l’animation d’une position à une autre. On passe direcetemet de la position A à une position B dont on définit l’angle en paramètre.  
register\_shape() peret d’enregistrer une forme déjà définie.

Clock\_face : Définit les bâtonnets sur l’image en arrière-plan de l’horloge.



Pensize() définit la taille des bâtonnets noirs sur le côté de l’horloge.  
La boucle "for i in range(60) :" définit la position des bâtonnets placés tous les 5 unités de temps, comme sur une horloge standard.

**Les problèmes que j’ai rencontré avec Python & Kivy**

Tout d’abord nous avons rencontré un premier problème lors de l’ébauche du code du chronomètre. En effet le chronomètre se décalait au fur et à mesure du temps.   
Pour le résoudre nous avons modifié l’arrondi, ce qui nous a permis d’avoir une mesure beaucoup plus précise.  
Ensuite nous avons constaté qu’en fonction de l’éditeur que nous utilisions, le programme ne mettait pas le même temps à tourner, ce qui modifiait notre mesure du temps. Pour contrer ce problème nous avons choisis de créer une boucle qui se lancerait automatiquement lors du lancement de l’application et qui calculerait le temps que met notre programme à tourner, ce qui permettrait ensuite de faire les ajustements nécessaires.  
Ensuite nous avons codé notre interface graphique. Nous avons d’abord utilisé une fonction du répertoire time comme un chronomètre, cependant la fonction prenait juste l’heure de l’ordinateur comme référence ce qui ne nous intéressait pas du tout ! Nous avons choisis de remplacer cette fonction par notre propre script (chronomètre).  
Un de nos problèmes majeurs est que la boucle servant de chronomètre se remettait à zéro, ce qui se ressentait sur l’interface graphique : notre aiguille oscillait entre 0 et 1. Pour parer ce problème nous avons modifié intégralement notre fonction, pour ne plus à avoir à définir à 0, à chaque début de boucle, nos variables.   
Cependant, le problème persiste malgré notre solution. En effet dès que nous lions notre interface graphique à notre chronomètre, qui fonctionnent chacun bien séparément, le chronomètre bug. Le chronomètre oscille entre 1 et 2 sans aucune raison apparente, l’aiguille est alors elle aussi bloquée entre ses positions 1 et 2.

**Intégration et validation**

Pour pouvoir avancer dans notre projet nous avons réalisé de nombreux tests intermédiaires. Les premiers types de tests consistaient juste à lancer le programme et à comprendre pourquoi il fonctionnait ou non. Si l’erreur était compliquée à trouver nous utilisions la méthode des print(). C’est-à-dire que nous écrivions de nombreuses commandes de print() tout au long du programme. Dès qu’une valeur du print() ne s’affichait pas nous savions que le programme n’avait pas réussi à tourner jusqu’à ce point. Nous avions donc réduit la zone de recherche de l’erreur. Nous pouvions donc ensuite localiser facilement la source de l’erreur.

De plus pour notre projet nous avons réalisés plusieurs tests sur différents téléphones, avec l’application sur Linux. Ces tests ont été plus ou moins concluants et consistaient juste à lancer le programme sur le téléphone.

Après qu’un test nous confrontait à une erreur technique, nous utilisions Internet pour comprendre l’erreur, notamment à l’aide du site Stack Overflow.

**Bilan et perspectives**

Notre projet de départ était ambitieux, et n’avait pas à vocation d’être entièrement finit. Notre projet de départ prévoyais trois fonctionnalités dans notre application :   
 - un chronomètre  
 - un « timer »  
 - une fonctionnalité sportive

Sur ces trois fonctionnalités nous n’avons gardés que les deux premières. Ensuite notre chronomètre ne fonctionne pas entièrement comme nous le souhaitons. En effet les différentes parties de notre programme marche séparément, mais pas ensemble. L’interface graphique de notre chronomètre ne se lie pas au code du chronomètre. L’aiguille censée se déplacer oscille entre 0 et 1.

Notre projet consistait aussi à décliner notre application sur smartphone. Nous avons réussi cette partie du projet sur des téléphones Androïd.

Les différentes améliorations possibles de notre projet :  
 - régler notre problème sur le chronomètre  
 - rendre les interfaces graphiques plus agréables à l’œil  
 - développer la dernière fonctionnalité sur le sport  
 - décliner le projet sur d’autres smartphones, comme ceux d’Apple ou de Microsoft.