**Projet d’ISN – Dossier**

**Automatisation d’une expérience de sciences cognitives sur les méthodes mnémotechniques**

*Zoé GUILLEMAIN – TS4  
(travail sans binôme)  
2016-2017*

**PRÉSENTATION DU PROJET**

**Problématique :** comment optimiser le déroulement d’une expérience scientifique grâce à une application ?

**But du programme :**  
J’ai étudié l’année dernière une expérience scientifique pour tester l’efficacité de la technique du Palais Mental (ou lieux de mémoire). L’expérience consiste à lire deux listes de mots à un groupe de sujets ; la première doit être retenue sans utiliser de moyen mnémotechnique, la deuxième en utilisant le Palais Mental : imaginer un lieu ou trajet bien connu, s’y promener mentalement et s’imaginer y déposer les objets de la liste. On a remarqué que le nombre de mots retenus avec la méthode était beaucoup plus élevé que sans la méthode.

J’ai mené le test sur ma classe, ce qui a posé certains problèmes d’organisation. Entre autre, la gestion du groupe, le risque de falsification des résultats si les sujets communiquent, l’incertitude liée à la vitesse de lecture des mots, la récupération et l’analyse des résultats… C’est pourquoi j’ai décidé d’automatiser le test pour régler ces problèmes :  
-chaque sujet peut faire le test indépendamment, au moment qui lui convient, sans qu’il y ai besoin de gérer un groupe turbulent  
-les mots sont lus à la même vitesse pour les deux listes et pour tous les sujets  
-les résultats sont enregistrés dans une base de donnée et traités sous forme d’histogramme par le programme, ils sont donc plus facilement appréhendables

**Cahier des charges:**  
Il faut faire passer deux tests, constitués chacuns de trois étapes: affichage des mots un par un à l’écran, à une fréquence raisonnable (un toutes les trois ou quatre secondes); une étape perturbatrice empêchant l’utilisation de la mémoire à court terme (calcul mental, animation…), et restitution des mots retenus, puis comparaison à la liste de mots affichés et enregistrement du score de restitution pour analyse ultérieure des résultats.. Pour les deux tests, la liste de mots et le contenu de l’étape perturbatrice doivent être différent.

Il faut enregistrer les résultats de chaque sujet, et les présenter individuellement et comparativement (par exemple sous forme d’histogramme).

**Compétences informatiques :**

*Dimension algorithmique :* création des listes aléatoires, affichage mot par mot, confrontation de listes, enregistrement des mots retenus, manipulation des données.   
*Eléments de programmation :* langage utilisé : Python ; IDE : Idle (en classe) et PyCharm (à la maison); utilisation du langage SQL pour la base de données (SQLite)  
*Représentation de l’information :* affichage des textes explicatifs, listes, zones de textes et boutons ; affichage des résultats sous forme de listes et d’histogrammes, succession d’écrans ➔ utilisation de la bibliothèque Tkinter

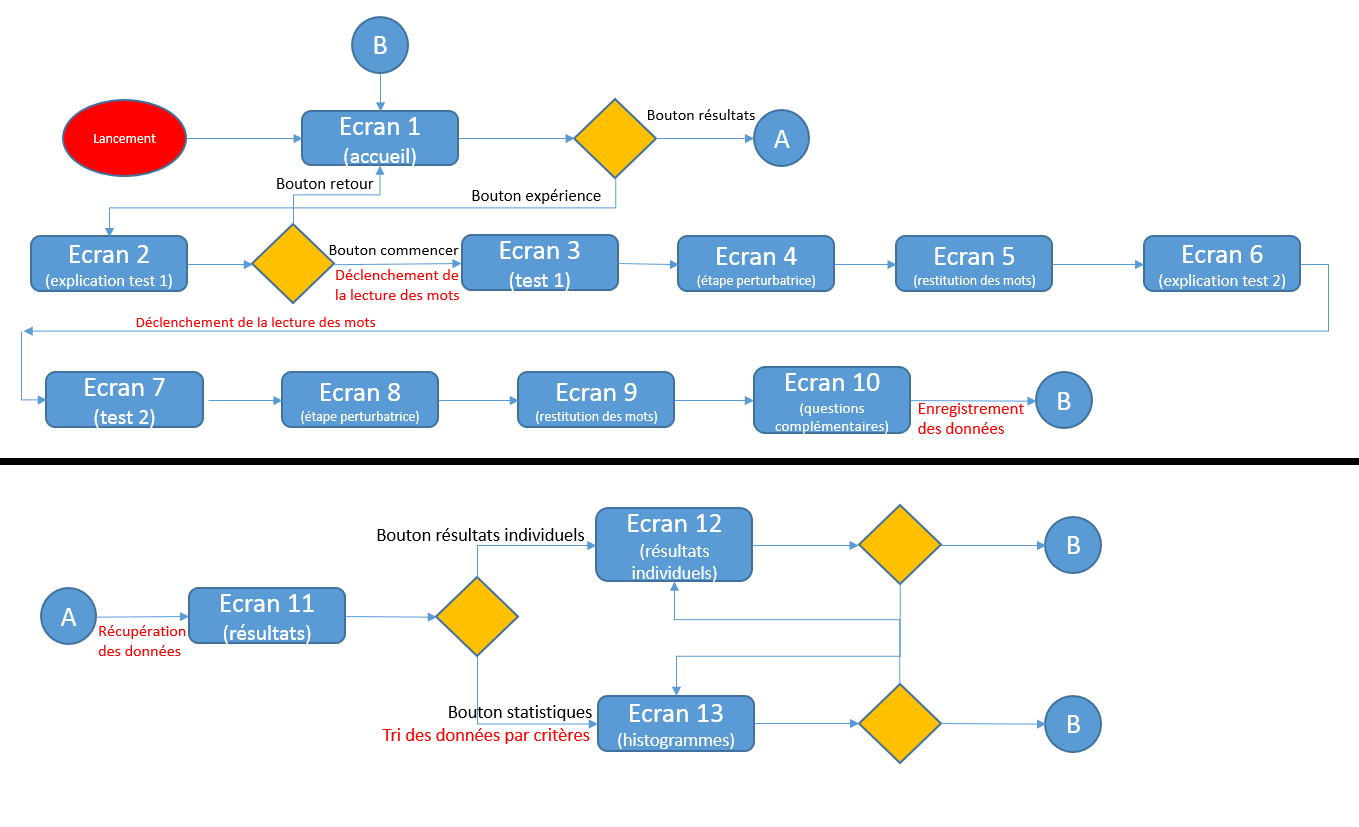
**Déroulement de l’expérience :**

Le premier écran propose deux options : passer le test, ou voir les résultats des tests précédents.

*Le test :* il y a d’abord un texte explicatif du déroulement. Puis une liste de dix mots est affichée à l’écran, chaque mot affiché puis effacés un par un.   
Ensuite, il y a une étape perturbatrice : on pose au sujet plusieurs questions de calcul mental (cela empêche qu’il utilise la mémoire à court terme pour retenir les mots). S’il répond mal à une question, elle est reposée.   
On passe ensuite à la restitution des mots. Le sujet doit entrer un par un tous les mots dont il se souvient.   
Puis il y a un nouveau texte explicatif, décrivant le moyen mnémotechnique du Palais Mental. Ensuite, les mêmes étapes recommencent (avec une liste de mots différents évidemment).   
Après les deux tests successifs, on annonce au sujet ses résultats aux deux tests, et on lui demande des informations supplémentaires : son âge et son sexe (ces questions sont facultatives ; si le sujet n’y répond pas, la valeur attribuée à ces deux données sera ‘’non renseigné ‘’).

*Les résultats :* on propose deux présentations des résultats.   
D’abord, sujet par sujet : pour chaque sujet on donne ses résultats aux deux tests, son sexe et son âge, et l’évolution du score.  
 Les résultats sont également présentés sous forme d’histogramme, avec le nombre mots retenus en abscisse, et pour chaque nombre de mots retenus la proportion de personnes au test 1 et au test 2.

**Organigramme de l’application**

****

**Fonctionnement du programme :**

Il y a une fenêtre principale, sur laquelle se superposent plusieurs cadres, un pour chaque étape de l’expérience. On crée tous les cadres dès le début du programme, mais ils sont affichés seulement lorsqu’on appuie sur le bouton qui les appelle.   
Dans le programme principal on définit les caractéristiques de chaque cadre : son nom, et les widgets qui sont dessus. Toutes les actions faites par le programme (affichage des mots, changement de cadre, enregistrement des données…) sont effectuées par des fonctions. Certaines fonctions (comme la création des listes) sont appelées dès l’ouverture du programme. Les fonctions gérant l’affichage sont toutes appelées par des boutons : comme la partie graphique (fenêtre, cadres) n’est effectuée qu’à la toute fin du programme (avec le .mainloop() ), on ne peut pas faire d’action dans le programme principal.

Certaines fonctions sont appelées par plusieurs boutons différents : par exemple changerecran, qui permet de passer d’un cadre au suivant. Les autres sont appelées par un ou deux boutons au plus, car il s’agit d’une action qui n’est effectuée qu’une seule fois dans tout le programme (ou deux pour l’affichage des mots, l’étape perturbatrice et la restitution des mots retenus, puisqu’il y a deux tests successifs). La plupart contiennent des boucles (par exemple l’affichage des mots, ou l’enregistrement des résultats dans la base de données).

**Contraintes :**

Avant de commencer le projet je savais que je devrais apprendre à utiliser Tkinter (nous avions vu la partie dessin graphique en classe mais pas les widgets), et les bases de données. Au final j’ai eu plus de choses à étudier car j’ai eu besoin de la méthode .after pour faire attendre des fonctions, des StringVar et IntVar pour Tkinter, et j’ai dû apprendre à faire interagir des fonctions entre elles et à gérer des variables globales.  
J’ai utilisé des tutoriels sur des sites et des forums pour me renseigner sur ces différents aspects. Avant d’intégrer quoi que ce soit au programme principal j’ai d’abord commencé par faire des petits exercices pour tester chaque méthode séparément, puis écrire le bout de programme nécessaire dans un fichier à part avant de l’intégrer au programme principal une fois qu’il fonctionnait comme je le souhaitais.

*Problème d’affichage :* pour chaque étape de l’expérience j’avais défini un écran Tkinter, qui était détruit lorsqu’on n’en avait plus besoin. Après avoir intégré au programme l’attente dans l’affichage des listes, j’ai été confrontée à un problème : IDLE effectuait tout le programme (dont les parties où on attend plusieurs secondes), PUIS affichait les écrans (puisque la commande fenetre.mainloop() se trouve à la fin).   
J’ai réglé ce problème en créant un seul écran, sur lequel se plaçaient successivement plusieurs cadres, un par étape de l’expérience. Tous les cadres sont créés lors de l’ouverture du programme, mais sont cachés. Pour passer d’un cadre à un autre, plutôt que de faire pack et destroy comme précédemment avec les fenêtres, les boutons de changement d’étape appellent une fonction qui fait cacher le cadre 1 et afficher le cadre 2 (avec pack et pack\_forget ). De plus, toutes les parties demandant au programme d’attendre sont maintenant dans des fonctions, qui sont appelées seulement par des boutons (comme ça le programme n’attend pas à des moments inopportuns).

*Problème d’attente :* après avoir modifié le programme pour avoir des cadres plutôt que des fenêtres, et mis les parties d’attente dans des fonctions, j’ai eu un nouveau problème : la méthode .sleep ne fonctionnait pas comme je m’y attendais, elle bloquait tout le programme de manière permanente plutôt que de le mettre en pause quelques secondes. J’ai donc dû apprendre la méthode .after, qui était plus adaptée au fonctionnement de mon programme, mais plus compliquée. En effet, j’avais du mal à savoir à quel objet l’intégrer, et je ne savais pas qu’il fallait ajouter un .update pour qu’elle soit effective. La partie d’affichage mot par mot des listes m’a donc pris beaucoup de temps, mais je ne pouvais pas simplifier cette partie et contourner le problème, car le fait que les mots soient lus un par un et pas en bloc est le centre même du projet.

*Problème de restitution :* la restitution des mots retenus via un widget entry a nécessité d’apprendre à utiliser les objets StringVar, que je ne connaissais pas. J’avais commencé par utiliser des variables string normales, mais cela posait problème: une fois le contenu de l’entry enregistré dans la variable, l’entry restait plein, et on ne pouvait pas y mettre une nouvelle valeur. Avec une StringVar, il y a un contrôle dans les deux sens: on entre une valeur dans l’entry, et elle est rentrée également dans la StringVar (et ensuite enregistrée dans une variable, puis dans la base de données). Ensuite, on vide la StringVar, et l’entry se vide aussi: on peut y rentrer une nouvelle valeur.

*Stockage des données :* cette partie ne m’a pas posé de problèmes majeurs, à part le fait que j’ai dû apprendre à me servir de la technologie SQLite. L’intérêt d’utiliser une base donnée pour stocker les résultats, plutôt qu’un simple tableau dans le programme, est qu’on conserve les données lorsque le programme est fermé. De plus, pour le traitement des résultats, on peut demander à récupérer des données répondant à certains critères (par exemple : les résultats de toutes les personnes de moins de 20 ans), et l’ordinateur les trouve tout seul ; plutôt que de devoir écrire soi-même une fonction servant à trier la base de donnée avant d’afficher les résultats. On peut également récupérer la moyenne ou la médiane d’une liste de valeurs (mais je n’ai pas eu besoin de cette option).  
J’ai également dû créer une fonction qui ajoutait des données “artificielles” à la base, ainsi on avait des données à visualiser même si personne n’avait encore passé le test (les données ajoutées sont celles de mon TPE de l’année dernière).

*Traitement des données:  
Résultats individuels:* j’avais d’abord affiché les résultats à l’aide d’une succession de labels, mais la fenêtre était trop petite pour afficher toutes les données. J’ai d’abord essayé de configurer une scrollbar, mais comme j’avais un ensemble de labels et pas un seul grand label, c’était impossible. J’ai donc changé de méthode et j’ai utilisé un widget listbox, qui a déjà une scrollbar intégrée. La mise en page des données adaptée aux labels, avec des sauts de ligne, n’étant pas prise en compte par la listbox, j’ai dû apprendre à utiliser le formatage de chaîne de caractères (j’ai dû le faire en tâtonnant, pour pouvoir aligner les colonnes).  
*Histogrammes :* pour tracer les histogrammes j’ai dû exprimer les coordonnées de chaque rectangle en fonction du résultat (nombre de mots retenus) et de l’effectif pour ce résultat. Je n’ai pas eu de problème avec l’ordonnée (l’effectif), mais j’ai eu du mal à trouver une manière d’exprimer les abscisses en fonction du nombre de mots retenus, sans avoir de décalage ou de rectangles superposés (puisque je devais mettre trois rectangles pour chaque résultat : le test 1, le test 2 et un espace blanc). J’ai fini par faire une boucle appelant une fonction qui traçait les trois rectangles à la suite, plutôt qu’un seul rectangle à la fois comme j’avais initialement prévu ; ainsi, les coordonnées du deuxième rectangle et de l’espace blanc dépendent de celles du premier rectangle, et pas des résultats seuls (ce qui aurait été ingérable).  
J’ai également ré-utilisé le formatage de chaînes de caractère pour que la légende de l’histogramme coïncide avec les colonnes.

**Organisation du temps de travail :**

J’ai travaillé majoritairement chez moi car cela me permettait de travailler pendant plus longtemps sans m’interrompre. J’ai commencé par préparer une petite présentation du projet que j’ai proposée à mon professeur : le but de l’application, l’organigramme présenté plus haut et une maquette de chaque écran.

Ensuite j’ai écrit les principales fonctions permettant de passer le test : la création des listes aléatoires, la lecture des listes, la restitution des mots retenus, la comparaison des mots retenus avec ceux lus, et l’enregistrement des résultats. Chaque fonction était écrite dans un fichier séparé. J’ai fait cela surtout pendant les vacances et les week-ends, et j’ai mis à profit les séances d’ISN pour demander de l’aide à mon professeur, rédiger des parties de code qui ne nécessitaient pas de recherche ou de brouillons, ou mettre en commun les différentes fonctions dans le programme principal et régler les bugs qui en résultaient.

Puis j’ai bâti le programme principal : j’ai mis toutes les fonctions dans le même fichier, téléchargé les modules (Tkinter, randrange, etc), et défini les écrans ; cette partie devait être faite après avoir créé les fonctions, puisque les boutons et labels appellent les fonctions. Néanmoins j’ai dû modifier les fonctions au fur et à mesure, pour les adapter au fonctionnement de Tkinter (ce que je n’avais pas pensé à faire dès le départ). La majorité des widgets sont définis dans le programme principal, bien que quelques-uns soient définis dans les fonctions.

J’ai fait la partie d’exploitation des résultats en dernier, car elle n’est pas nécessaire au fonctionnement du reste de l’application, et nécessite des compétences différentes de celles mises en œuvre pour la partie expérience : une base de donnée SQLite et les dessins graphiques de Tkinter. J’ai commencé par l’enregistrement et le stockage des données, avant de faire la partie exploitation, puisqu’elle nécessitait la présence de la base de données pour fonctionner.

Pendant les dernières séances de cours et quelques heures à la maison, j’ai réglé les derniers détails: vérifier que l’ensemble du test fonctionnait sans s’interrompre, ajuster la taille de la fenêtre, des frames et des widgets, soigner la mise en page des résultats individuels et de l’histogramme.  
Il restait quelques bugs: un problème de list out of range lors de l’étape perturbatrice (que j’ai réglé en lisant la liste dans l’ordre plutôt qu’en mode random); les résultats des tests qui n’étaient pas enregistrés (la variable nbretenu était initialisée à 0, mais elle n’était jamais changée; je l’ai remplacée par une StringVar qui était plus facilement mise à jour); les données “artificielles” qui n’étaient pas insérées dans la base de données (il s’agissait d’une simple erreur de syntaxe, j’avais oublié les parenthèses d’une commande qui du coup ne s’effectuait pas). Une fois que tout fonctionnait correctement, j’ai remis l’application en condition d’utilisation. Pendant la phase de développement pour gagner du temps, les mots étaient lus à raison de un par seconde et la liste de mots était réduite, je les ai réglé à un toutes les trois secondes. J’avais travaillé avec une liste de nombres en toute lettres, je les ai remplacé par des vrais mots et j’en ai rajouté pour avoir une liste plus grande. J’ai éliminé du code tous les test qui imprimaient un résultat dans l’interpréteur, ainsi que toutes les notes à moi même.

**Possibilités d’amélioration:**

Mon programme remplit actuellement sa fonction, à savoir faire passer le test de mémoire à des sujets, enregistrer les résultats et les exploiter. Néanmoins certains éléments pourraient être améliorés.

*Ergonomie*

Il serait mieux de pouvoir faire lire les mots à voix haute par l’ordinateur, ce que l’organisateur devait normalement faire. Comme Python ne gère pas les sons, j’ai du me contenter d’afficher les mots à l’écran.

Il faudrait écrire un algorithme pour reconnaître les mots restitués même s'ils sont légèrement différents des mots affichés: si l’utilisateur a fait une faute d’orthographe, c’est compté comme un mot non retenu, alors qu’il s’est souvenu du mot. Même chose pour la casse, si il a mis des majuscules ou des tirets. J’ai essayé de limiter le problème en ne proposant que des mots sans accent et sans tiret, avec une orthographe simple; mais le mieux serait d’écrire un algorithme qui vérifie si le mot restitué est à 80 ou 90 % identique au mot affiché.

Actuellement, l’utilisateur doit cliquer sur un bouton pour valider chaque entrée, il serait plus agréable que la touche “Enter” valide sa saisie

*Restitution des résultats de l’expérience*

J’avais initialement prévu de présenter plusieurs histogrammes, présentant les données en fonction de certains critères (âge, sexe, seulement les résultats du test 1, seulement ceux du test 2…). Je n’ai pas eu le temps de programmer le tri des données et les histogrammes correspondants, ce serait donc une amélioration possible.

*Organisation du code*

L’organisation du code peut aussi être améliorée sur certains aspects; j’ai parfois deux fonctions différentes pour la même étape des deux tests; je l’ai programmé de cette manière car c’était plus facile à manipuler, mais il serait mieux de les fusionner en une seule fonction. Si le code de ce programme devait être lu par d’autres programmeurs, il serait également utile que je me renseigne sur les conventions concernant les noms de variables et de fonctions: ici, j’ai mis au point ma propre notation (du type label\_ecran1, bouton\_ecran2, etc), pour pouvoir m’y retrouver dans mon code et savoir facilement à quel type d’objet j’avais affaire, mais ce n’est pas nécessairement la syntaxe utilisé par la majorité des programmeurs.

**Ressources documentaires**

J’ai utilisé des tutoriels sur Internet, en particulier les sites apprendre-python.com, stackoverflow.com, et le manuel de Python et de Tkinter.

Tkinter :  
<http://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>  
<http://fsincere.free.fr/isn/python/cours_python_tkinter.php>

Gestion de temps :  
<http://stackoverflow.com/questions/25753632/tkinter-how-to-use-after-method>

SQLite :  
<http://apprendre-python.com/page-database-data-base-donnees-query-sql-mysql-postgre-sqlite>  
<http://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html>  
<http://www.sqlfacile.com/apprendre_bases_de_donnees/count_et_group_by_1>

J’ai également reçu de l’aide de mon professeur d’ISN, Monsieur Tamby, en particulier pour le problème d’attente du programme (la méthode .after) et pour la création de l‘histogramme ; et de ma mère informaticienne pour gérer la base de données SQLite, découvrir l’IDE PyCharm, et pour l’organisation générale du programme principal et l’optimisation des méthodes de programmation.

**Analyse de la dimension sociétale**

Ce programme est à but scientifique. Son objectif est d’automatiser l’expérience pour pouvoir la faire passer à plus de gens, et donc avoir plus de résultats à analyser. Dans le cadre d’une utilisation professionnelle, cela permettrait d’étudier l’impact des moyens mnémotechniques sur un large panel de sujets, en limitant les coûts et en fiabilisant les résultats. Cela peut donc avoir un intérêt économique.

Il n’atteint pas à la sécurité ou la vie privée des sujets, puisqu’aucune information personnelle (nom, adresse...) n’est retenue, et que les données supplémentaires (âge et sexe) sont facultatives. Il n’y a pas non plus de question de droit d’auteur, puisqu’aucune image, texte ou autre provenant d’Internet n’est utilisée. De même pour le code: j’ai essayé de limiter le copier-collé, préférant réécrire manuellement chaque élément, pour les adapter à mes besoins et me familiariser avec les méthodes. De plus, tous les tutoriels et exemples du Net dont je me suis inspirée étaient libres de droit, puisqu’il s’agissait pour la plupart de forums (dont le but même était de partager son code avec d’autres programmeurs), ou du manuel officiel de Python.

**Auto-évaluation finale individuelle**

J’ai travaillé seule sur ce projet, il n’y a donc pas eu d’aspect gestion d’équipe/des compétences .J’ai appris à utiliser différents outils de Python, en particulier l’interaction graphique avec l’utilisateur à travers de Tkinter, et la gestion de données avec SQLite. Mais j’ai surtout développé une méthode de travail en informatique, car mener un projet long comme celui-ci demande plus d’organisation et de planification que pour les TDs faits en classe. J’ai par exemple appris à tester toutes les fonctions indépendamment avant de les ajouter au programme principal, à gérer des fonctions qui s’appellent les unes les autres, à partager des variables globales entre plusieurs parties du programme, à faire des tests à intervalle régulier…J’ai également appris à optimiser les fonctions après avoir écrit une première version fonctionnelle. J’ai donc appris autant sur le plan technique, spécifique à Python, que sur le plan méthode, important pour la programmation en général.

Ce projet m’a conforté dans mon envie de poursuivre mes études en tant qu’ingénieur dans l’informatique.