Minis projets

Pour l'ensemble des projets, nous pourront utiliser n'importe quel langage de programmation.

> Chaque projet avec un 📄 possède un code d'exemple d'implémentation basique en **python** est disponible à l'adresse https://github.com/BlooSkyd/Minis-projets

Conjecture de Syracuse 📄

Il s'agit d'une fonction très simple, qui permet de retomber toujours sur le même résultat :

```
Fonction:   Syracuse
Paramètre:  n
Retour:     Si n pair : n/2
            Sinon : 3*n + 1
Fin: Dès que le résultat vaut 1
```

Créer un code qui demande à l'utilisateur le nombre qu'il souhaite et affiche chaque étape de la fonction syracuse jusqu'à ce que le résultat vale 1.

⚠ Attention à bien prendre en compte la condition d'arrêt.

Jeu du motus 📄

L'idée est de faire un petit programme qui choisi un mot, et nous l'affiche de manière "*sensurée*".
L'utilisateur devra donc saisir des mots comme tentatives, et les lettres communes seront affichées.
L'affichage de retour est à déterminer, soit avec des caractères spéciaux (?,#,€, etc) ou bien via des couleurs de manière plus proche du jeu existant.
Nous pourrions avoir une évolution avec un système de vie.

Exemple :

- Mot à trouver : `bateaux`
- Affichage : `------`
- Saisie : `palmier`
- Réponse : `-a---?-` (affichage relatif à la saisie)
- Saisie : `tambour`
- Réponse : `?a-?-u-` (affichage relatif à la saisie)

1. Réaliser le programme pour gérer un cas d'usage
2. Améliorer le programme pour avoir un système de vie
3. Améliorer le programme pour avoir une liste générée aléatoirement / même pas connue du développeur (bibliothèque tel que Faker?), et donner un résumé de la saisie (nombre de lettres bien placées, mal placées, etc)

Code César 📄

Connaître le code césar, c'est mettre un pied dans l'univers du cryptage et de la cybersécurité (un tout petit pas, mais un pas quand même)

Son fonctionnement est très simple : pour une lettre donnée, il renvoie une autre lettre de l'alphabet avec un décalage donné.
Par exemple, avec un décalage de 3 : E donne H et L donne O.

On peut aborder le projet de manière croissante :

1. D'abord commençons par crypter le message :
    - Réalisons une fonction qui pour une lettre donné et une valeur de décalage, renvoie la nouvelle lettre.
    - Réalisons une fonction qui prend une chaine de caractère (un message) et une valeur de décalage, et qui renvoie le nouveau message
2. Ensuite, décryptons le message :
    - Réalisons une fonctione qui prend une chaine de caractère et une valeur de décalage, et renvoie le message décodé (est-ce qu'on a vraiment besoin de le faire ?)
3. Enfin, suggerons des propositions :

    La langue française utilise les 26 lettres et le E est celle la plus utilisée.
    Réalisons une fonction qui prend une chaine de caractère et renvoie quel caractère est le plus présent

    - Réalisons une fonction qui détermine l'écart dans l'alphabet entre deux caractère (a => a = 0, a => e = 4, e => a = -4)
    - Réalisons une fonction qui prend une chaine de caractère et renvoie le message déterminé comme étant le plus probable d'être le bon
    - Réalisons une fonction qui prendre une chaine de caractère et une valeur de décalage, et qui renvoie le message converti avec le décalage fourni ET le message le plus probable

Jeu du pendu 📄

Qui ne connait pas le jeu du pendu ?
Au cas, petite présentation : un mot censuré et il faut le retrouver.
Pour cela, les joueurs proposent des lettres une a une.
Si la lettre compose le mot recherché, toutes les occurences sont affichées.
Sinon, on perd une vie, symbolisée par la progression d'un dessin de scène d'éxécution par pendaison (un peu gloque oui).

Selon les variantes, les joueurs aiment bien avoir la première lettre de dévoilée dès le début (et toutes ses occurences) ainsi que d'affichier les lettres déjà proposées.
Également selon les versions, le dessin évolue de manière à dessiner la structure ou et le bonhomme, ou bien seulement le bonhomme (moins d'essais possibles).

Pour faciliter l'affichage, voici un modèle du dessin complet :

```
dessin = "[]=======v=\n"\
         "   |/    |\n"\
         "   |     O\n"\
         "   |    /|\\\n"\
         "   |    / \\\n"\
         "   |\n"\
         "=============";
```

On pourra réutiliser des fonctionnements issus du jeu du motus programmé précédemment.

Décomposé étape par étape, nous pouvons :

1. Créer une fonction `devoiler()` qui prend en paramètre le mot à trouver, l'avancée du mot à trouver, une lettre et qui renvoie la nouvelle avancée du mot à trouver.

   > Exemple : `devoiler("toboguant", "t---g--nt, "o")` renvoiera `"to-og--nt"`

2. Créer une fonction qui affiche les lettres déjà saisie précédemment
3. Créer une fonction qui affiche l'état du dessin et le fait évoluer en cas d'erreur
4. Gérer l'ensemble du code pour que le jeu fonctionne du début à la fin

Idem, nous pourrons améliorer le projet en l'agrémentant d'une liste générée aléatoirement.

Jeu du morpion 📄

Pour créer un morpion assez facilement, il faut commencer par le plateau : un tableau à une dimension de 9 cellules.

Nous pouvons faire une fonction d'affichage légèrement stylisée afin d'avoir un plateau de 3 cases de haut sur 3 cases de large.

Ensuite, il nous suffira de demander au joueur d'indiquer dans quelle case il souhaite jouer, la gestion du caractère se fera par la suite.

> ⚠ Attention à la conversion entre la valeur de la case côté humain vs. côté machine, mais aussi à ce que la case ne soit pas déjà utilisée.

Si ces conditions sont bonnes, on sauvegarde le coup, on vérifie qu'il n'y a pas de victoire et qu'il ne s'agisse pas d'une égalité (9 coups max).

> Question : y'a t-il un ordre à privilégier dans la réalisation des tests ?

Si jamais il n'y a ni victoire, ni égalité, on change de joueur et on recommence.

On pourra par la suite réfléchir à la possibilité de jouer contre la machine, qui jouera d'abord de manière aléatoire et par la suite pourquoi pas se documenter sur quels sont les meilleurs coups à jouer en fonction de la situation.

Jeu du black jack

Parmis les jeux de carte de paris (on ne fait pas la promotion du paris, jouer comporte des risqutes, blabla bla, etc.), le jeu du black jack est l'un des plus simples à concevoir.

Pour le présenter rapidement, il consiste en une partie opposant le croupier (le représentant du casino) aussi appeler la banque, et les différents joueurs. Dans notre cas, nous jouerons en 1 contre 1 et le croupier sera un simple automate. Nous ne nous occuperons pas de la partie de mises (paris) et nous nous concentrerons uniquement sur la partie tirage et du jeu en tant que tel.

> Le but est de battre le croupier sans dépasser le score de 21. Dès qu'un joueur ait plus que 21, on dit qu'il « brûle » et il perd sa mise initiale. Wikipédia

La valeur des cartes sont les suivantes :

- de 2 à 9 : valeur nominale de la carte ;
- de 10 au roi (surnommées « bûche ») : 10 points ;
- as : 1 ou 11 points (au choix du joueur).

Un blackjack est la situation où le joueur reçoit, dès le début du jeu, un as et une bûche. Si le joueur atteint 21 points avec plus de deux cartes, on compte 21 points et non blackjack.

Réalisation

1. Construire les données nécessaires pour les cartes (réflexion à faire sur le type de données à utiliser)
2. Créer une fonction qui permet de piocher une carte et la renvoie
3. Gérer l'inventaire du joueur et du croupier
4. Initialiser la partie
5. Créer le code nécessaire pour gérer les intéractions avec les joueurs
6. Créer l'algorithme pour gérer le croupier
7. Gérer victoire et défaite

Algorithmes de tri

Pour l'ensemble des algorithmes produits, il sera plus facile de les réaliser en ayant une compréhension visuelle de leur fonctionnement. Nous vous invitons à consulter leurs pages Wikipédia respectives.

Un résultat de fonctionnement visuel étape par étape est attendu.

Tri à bulle : ☆

Il s'agit d'un mode de tri très imagé : imaginons que nos éléments soit dans un verre d'eau, et qu'ils sont tous rattachés à une bulle.
Si l'élément est lourd, la bulle reste en bas, sinon la bulle remonte.
Sauf qu'en programmation, on préfère généralement avoir une structure triée de manière croissante, donc pour respecter cela on fera l'inverse :
Les éléments lourds remontent, et les légers descendent.

Le concept fort du tri à bulle, est qu'il fonctionne en comparant les éléments deux à deux, et fait remonter le plus grand.

Tri par insertion (par sélection) : ☆☆

C'est un tri très naturel, pratiquement le même que l'on fait lorsque l'on range des cartes dans notre main

On prend les deux premières cartes de gauche, on les compare, et on met la plus petite à gauche.
Puis, avec la 3e carte, on la compare à la 2e : si la 3e est plus petite, on la compare à la 1ère ; si elle est plus grande, on a fini.
Répéter jusqu'à la carte la plus à droite, et c'est fini !

Tri (par) fusion : ☆☆☆

C'est l'application du célèbre "diviser pour mieux reigner". À partir de deux listes triées, on peut facilement construire une liste triée comportant les éléments issus de ces deux listes (leur fusion).

Le principe de l'algorithme de tri fusion repose sur cette observation : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste (en fait, le plus petit des deux, à supposer qu'aucune des deux listes ne soit vide, sinon la réponse est immédiate).

L'algorithme est naturellement décrit de façon **récursive** :

1. Si le tableau n'a qu'un élément, il est déjà trié.
2. Sinon, séparer le tableau en deux parties à peu près égales.
3. Trier récursivement les deux parties avec l'algorithme du tri fusion.
4. Fusionner les deux tableaux triés en un seul tableau trié.

Tours de Hanoï

Les tours de Hanoï sont un vieux problème/jeu de réflexion mathématique. Il consiste à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

1. on ne peut déplacer plus d'un disque à la fois ;
2. on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ.

Pour ce programme, assurez-vous de comprendre le fonctionnement des structures FIFO et LIFO.

Réalisez le code nécessaire pour pouvoir réussir ce problème mathématique, d'abord avec 4 disques pour tester, puis 8 une fois que le code fonctionne. Conseils d'organisation :

1. Utiliser 3 listes avec pour chaque disque un entier de 1 à 8 ;
2. Créer une fonction qui affiche les 3 tours côte à côte ;
3. Créer une fonction qui prend en paramètre le numéro de la tour A, dans laquelle nous prendrons le disque et que nous poserons sur la tour B. Attention aux conditions de faisabilité ;
4. Intégrer cette fonction dans un algorithme demandant à l'utilisateur de saisir ces choix ;
5. Gérer les coups impossibles/victoires ;

Jeu des piles (à trier)

Nous faisons ici référence aux jeux mobiles assez populaires ces derniers temps, dans lesquels nous avons plusieurs piles avec des disques superposés et nous avons pour objectif de les triés par couleur.

Les piles ont un nombre restreint d'emplacement et nous ne pouvons mettre un disque que sur une pile vide ou un autre disque de la même couleur. Lors d'un déplacement d'une pile à une autre, tous les disques superposés les uns sur les autres et de la même couleur sont déplacés avec le premier, dans la limite de la capacité de réception de la pile d'arrivée.

Nous pourrons commencer par modéliser 5 piles avec des disques de 4 couleurs différentes. Ces piles auront une capacité maximal de 4 disques.

Conseils d'organisation :

1. Créer une fonction de création des piles, prennant en paramètres le nombre `n` de piles à créer et la capacité maximale `cap` de ces piles ;
2. Créer une fonction qui initiera les piles `p` créées en les prennant en paramètre ainsi que le nombre de couleur `ncol` différente. Cette fonction devra retourner les `n` premières piles -1 complétées aléatoirement des disques des `ncol` couleurs (en quantité exacte) et laisser 1 à 2 piles vides ;
3. Implémenter la ou les fonctions qui prendront une pile de départ et une pile d'arrivée et qui déplaceront l'ensemble des disques possibles, de manière assez similaire à ce qui a été fait pour l'exercice des Tours d'Hanoï ;
4. Vérifiez les conditions de victoire et de défaite (~ blocage).

Calculatrice

Le but est de pouvoir saisir des opérations plus ou moins simples et d'obtenir le résultat.
Pour cela, il faudra analyser les données passées en entrées et les décomposer en opération ou expression.

Exemples :

- Saisie : `5-2`
- Réponse : `3`
- Saisie : `3*3+6/2`
- Réponse : `12`

Pour cela, je propose de "simplement" décomposer chaque opérations en *expressions* :

1. e --> n (valeur)
2. e + e --> (expression)
3. e - e --> (expression)
4. e * e --> (expression)
5. e / e --> (expression)
6. e % e --> (expression)
7. (e) --> (expression)

Discord / Site web hébergé

La plupart des programmes précédents devraient pouvoir être hébergé sur un bot Discord, sur un site web aussi à condition de les convertir en javascript et de revoir un peu l'affichage.

Il ne reste plus qu'à le faire !

Le codage d'une application web (surtout dans le cas d'un bot Discord) impliqué la maitrise de quelques notions qui peuvent être globalement interprétées à partir de la page de documentation et des exemples données : https://discordpy.readthedocs.io/en/stable