1 Le cahier des charges

1. Les groupes de 2 ou 3 sont imposés.
2. Le choix des sujets est contraint :
   - une liste de sujets est proposée, tout sujet en dehors de la liste doit être validée par le professeur qui peut le refuser ;
   - deux groupes ne peuvent choisir le même sujet ;
   - les sujets faciles et guidés de la liste sont réservés aux élèves les moins à l'aise ;
   - le choix d'un sujet difficile sera valorisé.
3. Le rendu doit s'effectuer sous deux formes, au plus tard le jeudi 23/05 :
   - Un **rendu collectif** (un seul par groupe de projet) :
     - Sur **6 points :** dont 2 points évaluant la difficulté du sujet le code du projet et tous les fichiers ressources dans une archive zip nommée `projet-final-eleve1-eleve2-eleve3.zip`.
   - Un **rendu individuel** :
     - Sur **4 points :** sous la forme d'une capsule vidéo de 4 minutes minimum à 8 minutes maximum au format `mkv` ou `mp4`. Vous pouvez utiliser Vokoscreen : https://linuxecke.volkoh.de/vokoscreen/vokoscreen-download.html.
       La capsule doit être structurée en parties :
       * Une introduction avec présentation des objectifs du projet.
       * Une partie sur l'organisation du travail au sein du groupe (répartition des tâches, planification)
       * Une partie expliquant un extrait du code
       * Une partie libre sur un thème lié au sujet : histoire ou culture informatique
       * Une conclusion sur ce que vous avez appris ou développé comme compétences dans ce projet.

2 Quelques règles à respecter

- **Règle n°1** **Avant de coder je réfléchis crayon en main à l'architecture de mon programme et j'essaie de bien distinguer les données des fonctionnalités.**.
- **Règle n°2** **J'applique le principe de programmation modulaire :« *Chaque fois qu'une tâche peut être clairement séparée dans un programme, je l'encapsule dans une fonction et chaque fonction peut être écrite par une personne distincte.* ».** .
- **Règle n°3** **J'écris des tests unitaires pour chaque fonction de mon programme.**
- **Règle n°4** **Je prête attention à la lisibilité de mon programme : noms de variables et de fonctions explicites, autodocumentation par une docstring pour chaque fonction et commentaires pertinents des points les moins lisibles.**
- **Règle n°5** **Je prête attention à la portée des variables et j'essaie d'éviter l'usage de variables globales dans les fonctions.**

- **Règle n°6** **J'inclus un petit code client dans mon programme afin que le professeur puisse facilement le tester.**

- **Règle n°7** **Je prête attention à la portabilité de mon programme : j'utilise un encodage en UTF-8 et j'insère la directive `# -*- coding: utf-8 -*-` au début de mon script.**

- **Règle n°8** **Je ne perds jamais de vue mes objectifs, je fais un point régulier avec les membres de mon trinôme et les professeurs , j'utilise des outils de travail collaboratif.**

3 Liste de projets

3.1 Cinq Sujets guidés

Une archive ressources-pf-2024.zip est fournie avec les ressources (images et fichiers textes) nécessaires pour traiter ces sujets guidés.

3.1.1 Sujet 1 : programmation d'un jeu de pendu (*facile*)

- L'ordinateur choisit aléatoirement un mot dans le fichier `dicoSansAccents.txt` fourni.
- Le joueur doit deviner le mot en un nombre fixé de tours, à chaque tour il peut choisir entre la proposition d'une lettre ou du mot complet. Si le lettre proposée se trouve dans le mot l'ordinateur affiche le mot avec les lettres trouvées ou des tirets bas à la place des lettres non découvertes. Si le lettre a déjà été proposée ou si elle n'appartient pas au mot, l'ordinateur le signale au joueur. La boucle se termine si le joueur a proposé le mot ou si le nombre de tours maximal est atteint.
- A la fin de la partie, l'ordinateur affiche un message de conclusion.
- Proposer une petite interface graphique avec nsi_ui fourni dans le manuel Hachette (pages 53 et 54).

3.1.2 Sujet 2 : simulation en console d'un jeu de 2048 (*moyen*)

Le 2048 est un jeu vidéo développé par Gabriele Cirulli en 2014. Le principe inspiré de celui du taquin consiste à faire glisser des tuiles numériques sur une grille carré de 4×4 cases jusqu'à obtenir le nombre 2048. Par exemple si on déplace les tuiles vers la droite, elles glissent toutes horizontalement si des cases vides le permettent et deux cases adjacentes sur la même ligne et portant le même nombre se combinent en une seule case portant le double du nombre. Si on déplace les tuiles vers la gauche ou verticalement vers le bas ou vers le haut, on peut de même combiner des cases sur une même ligne ou une même colonne. Avant chaque mouvement un 2 ou un 4 apparaît aléatoirement sur l'une des cases vides.
L'objectif de ce projet est de développer un jeu de 2048 qui s'exécute dans la console `Python`. On donne ci-dessous une capture d'écran d'un début de partie. La grille est affichée avant chaque mouvement avec le 2 ou le 4 apparu aléatoirement dans l'une des cases vides après le mouvement précédent.

```
In [*]: 1 jeu_2048()
|  0  |  0  |  0  |  0  |
|  0  |  0  |  0  |  0  |
|  0  |  0  |  0  |  4  |
|  0  |  0  |  0  |  0  |
--------------------
Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?  4
|  0  |  0  |  0  |  0  |
|  0  |  4  |  0  |  0  |
|  4  |  0  |  0  |  0  |
|  0  |  0  |  0  |  0  |
--------------------
Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?  9
|  0  |  0  |  0  |  0  |
|  0  |  0  |  0  |  0  |
|  0  |  2  |  0  |  0  |
|  4  |  4  |  0  |  0  |
--------------------
Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?  4
|  0  |  0  |  0  |  0  |
|  0  |  0  |  2  |  0  |
|  2  |  0  |  0  |  0  |
|  8  |  0  |  0  |  0  |
--------------------

Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?
```

On s'appuiera sur l'énoncé fourni dans le document `DM-2048V1.pdf` inclus dans ressources.zip.

3.1.3 Sujet 3 : chiffrements de César et Vigenère (*moyen*)

- La **cryptographie** (du grec *kruptos* : caché, et **graphein** : écrire) qui est l'art de coder le message d'une façon connue uniquement de l'émetteur et du récepteur. On appelle **chiffre** un procédé de cryptage d'un texte caractère par caractère.

- D'après la légende, César aurait chiffré sa correspondance avec un **chiffre par substitution monoalphabétique** : chaque lettre de l'alphabet est remplacée dans le texte chiffré par une autre lettre, toujours la même. Il existe aussi des **chiffres par substitution polyalphabétique** comme le chiffre de Vigenère : chaque lettre de l'alphabet est remplacée dans le texte chiffré par une autre lettre, mais qui varie selon la position dans le message.

  Dans le chiffre de César, chaque lettre du texte chiffré s'obtient par un décalage de la lettre du texte en clair. Ce décalage est la clef du chiffre, pour le chiffre de César cette clef est 3 : A est chiffré par D, B par E,W par Z et X par A.

  Notre alphabet comptant 26 lettres, on peut repérer A par 0, B par 1 ... Z par 25.

  Le chiffre de César peut alors se modéliser sous la forme d'une fonction mathématique qui à une lettre en clair repérée par x avec $0 \leqslant x \leqslant 25$ associe une lettre chiffrée repérée par $y \equiv x+3 \mod 26$.

  Cette notation se lit y congru à $x+3$ modulo 26 et signifie que y est égal au reste de la division euclidienne de $x+3$ par 26. On peut changer la clef de décalage et si on prend 13 on obtient le chiffrement `rot13` déjà rencontré.

Lettre en clair	A	B	...	W	X	Y	Z
x	0	1	...	22	23	24	25
$y \equiv x+3 \mod 26$	3	4	...	25	0	1	2
Lettre chiffrée	D	E	...	Z	A	B	C

- Pour déchiffrer un message codé par le chiffre de César, si le message comporte suffisamment de caractères, on peut procéder par *analyse fréquentielle*.

  On recherche la clef de déchiffrement, entre 0 et 25, qui minimise la distance entre l'histogramme du texte déchiffré et l'histogramme de répartition des lettres dans la langue ciblée.

  Par exemple, en Français, la liste des fréquences des lettres minuscules de `'a'` d'indice 0 à `'z'` d'indice 25 est :

```
freqref = [8.4, 1.06, 3.03, 4.18, 17.26, 1.12, 1.27, 0.92, 7.34, 0.31,
    0.05, 6.01, 2.96, 7.13, 5.26, 3.01,0.99, 6.55, 8.08, 7.07, 5.74,
    1.32, 0.04, 0.45, 0.3, 0.12]
```

- Le chiffre de Vigenère est un **chiffre de substitution polyalphabétique** : chaque lettre en clair est remplacée par une lettre chiffrée mais cette traduction varie selon la position de la lettre en clair dans le message.

  Il s'agit d'un chiffre de César dont le décalage change selon la position de la lettre en clair dans le message.

  Par exemple, soit le texte en clair IL FAIT BEAU et la clef PLUIE.

  Sur la première ligne on écrit le texte en clair sans espaces, sur la deuxième on répète la clef sur la même longeur, sur la troisième on écrit le texte chiffré :

  ILFAITBEAU

  PLUIEPLUIE

  XWZIMIMYIY

  Pour chaque lettre en clair, le rang de la lettre de la clef en dessous donne le décalage qu'on applique pour déterminer la lettre chiffrée. Les rangs varient de 0 pour A à 25 pour Z.

  P de rang 15 en dessous de I donc I décalée de 15 donne X

  L de rang 11 en dessous de L donc L décalée de 11 donne W

  U de rang 20 en dessous de F donc F décalée de 20 donne Z ...

Voici le cahier des charges :

➟ Compléter le code de la fonction `chiffreCesar(source, decalage)` ci-dessous qui prend en argument une chaîne de caractères `source` et `decalage` un entier entre 0 et 25. La chaîne `source` ne doit pas comporter de caractères diacritiques, elle est convertie d'abord en majuscules, puis une chaîne `sortie` est construite en remplaçant chaque caractère alphabétique par son image par le chiffre de César de clef `decalage`.

```
def chiffreCesar(source, decalage):
    source = source.upper()
    sortie = ''
    for c in source:
        if c.isalpha():
            #à  compléter
        else:
            sortie += c
    return sortie
```

- Écrire une fonction `dechiffreCesarClef(source, decalage)` où `decalage` est la clef de chiffrement utilisée.

```
In [10]: chiffreCesar('Un python est un serpent constricteur.', 1)
Out[10]: 'BU WFAOVU LZA BU ZLYWLUA JVUZAYPJALBY.'

In [11]: dechiffreCesarClef('BU WFAOVU LZA BU ZLYWLUA JVUZAYPJALBY.', 1)
Out[11]: 'GZ BKFTAZ QEF GZ EQDBQZF OAZEFDUOFQGD.'
```

- Écrire une fonction `calculerFrequence(source)` qui retourne l'histogramme des fréquences (en %) des 26 caractères alphabétiques, dans une chaîne de caractères `source` composée de caractères non accentués et en majuscules.

- Écrire une fonction `calculerDistance(freq1, freq2)` qui calcule la distance euclidienne entre deux listes d'histogrammes de fréquences des 26 caractères alphabétiques.

  Si on note $(f_i)_{0\leqslant i\leqslant 25}$ et $(g_i)_{0\leqslant i\leqslant 25}$ les séries de fréquences des deux histogrammes, la distance euclidienne entre les deux histogrammes est égale à :

$$\sqrt{(f_0-g_0)^2+(f_1-g_1)^2+\cdots+(f_{25}-g_{25})^2}$$

- Écrire une fonction `calculerDecalage(source, freqref)`, qui prend en argument une chaîne de caractères `source` et une liste `freqref` représentant l'histogramme des caractères alphabétiques dans la langue ciblée (voir ci-dessus pour le Français). Cette fonction doit retourner le décalage entre 0 et 26 pour lequel l'histogramme de l'image de `source` par le chiffre de César présente une distance minimale avec `freqref`.

  En déduire une fonction `dechiffreCesar(source, freqref)` qui déchiffre un message crypté par le chiffre de César.

  Tester ces fonctions avec le contenu du fichier texte `clair.txt` qui contient le début du roman « *Les trois mousquetaires* ».

```
In [34]: f = open('clair.txt','r')

In [35]: message = f.read().upper()

In [36]: chiffre = chiffreCesar(message, 3)

In [38]: dechiffreCesar(chiffre, freqref) == message
Out[38]: True
```

- Écrire une fonction `chiffreVigenere(source, clef)` qui chiffre une chaîne de caractère `source` avec le chiffre de Vigenère pour une `clef` donnée.

  En déduire une fonction `chiffreVigenereClef(source, clef)` qui déchiffre une chaîne de caractères cryptée avec le chiffre de Vigenère pour une `clef` donnée.

```
In [62]: chiffreVigenere('AMSTERDAM', 'ROME')
Out[62]: 'RAEXVFPED'

In [63]: dechiffreVigenereClef('RAEXVFPED', 'ROME')
Out[63]: 'AMSTERDAM'
```

➟ Proposer une petite interface graphique qui permette de chiffre ou déchiffre un texte saisi par l'utilisateur avec le chiffre de César ou celui de Vigenère. On utilisera le module nsi_ui fourni dans le manuel Hachette (pages 53 et 54) et dans ressources.zip.

3.1.4 Sujet 4 : simulateur et solveur du jeu Sutom (*difficile*)

L'objectif de ce projet est de réaliser un simulateur du jeu sutom version française du jeu Wordle lui-même inspiré du jeu télévisé français motus.
Vous pouvez suivre les indications et compléter les squelettes de code fournis dans ressources-pf-2024.zip.

3.1.5 Sujet 5 : traitement d'images au format PGM/PBM/PPM *(facile)*

Les formats d'images PGM/PBM/PPM sont des formats d'images qui sont des fichiers textes où chaque pixel est décrit par une valeur. L'en-tête du fichier fixe des paramètres comme la largeur, la hauteur et le type pixel : noir et blanc, en niveaux de gris ou couleur (R,G,B).
Traiter les questions du document `TraitementImage.pdf` inclus dans ressources-pf-2024.zip.

3.2 Sujets non guidés

- **Automate cellulaire élémentaire à une dimension de Wolfram** ⇒ *Facile*

  – Lire la page Wikipedia :
  `https://en.wikipedia.org/wiki/Elementary_cellular_automaton`
  – Résoudre ce problème :
  `https://www.hackinscience.org/exercises/elementary-cellular-automaton`
  – Proposer une interface graphique minimale avec Tkinter qui prend en entrée un entier entre 0 et 255 et afficher une grille 79 × 40 avec les 40 premières générations à partir d'une ligne comportant un seul 1 médian comme dans
  `https://www.hackinscience.org/exercises/elementary-cellular-automaton`.

- **Jeu pixel art : pong, snake, angry birds, space invaders . . .** ⇒ *Moyen à difficile*

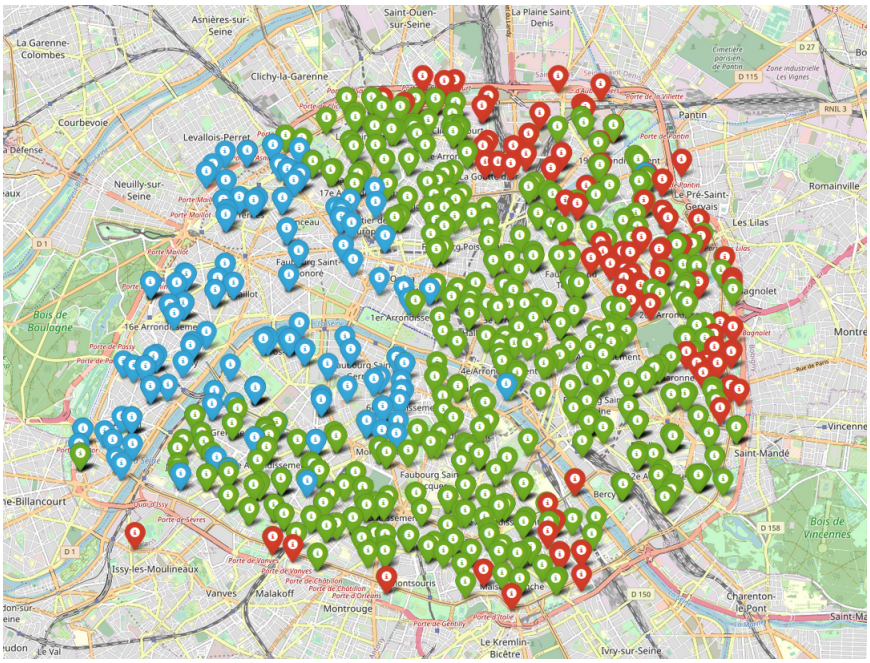
  Écrire une version Python du jeu Pong (ou Angry Birds, Space Invaders, Snake, Tetris . . .) On utilisera le module pyxel de Python.

- **Jeu de données ouvert et cartographie** ⇒ *Moyen*

  Récupérer sur une plateforme de *données ouvertes* (**open data**) deux jeux de données :

  – les résultats d'une élection par bureau de vote ;
  – les coordonnées géographiques de ces bureaux de vote.

  Avec le module folium de Python, marquer les bureaux de vote sur un fonds de carte avec des couleurs différentes selon les candidats arrivés en tête. On donne ci-dessous une carte représentant les résultats au premier tour des l'élection présidentielle 2017 dans les bureaux de vote parisiens.



Pour la carte précédente, on a utilisé des jeux de données ouverts disponibles sur https://opendata.paris.fr :

- résultats du premier tour de l'élection présidentielle 2017 :
  https://opendata.paris.fr/explore/dataset/elections-presidentielles-2017-1ertour/information/?disjunctive.id_bvote&disjunctive.num_circ&disjunctive.num_quartier&disjunctive.num_arrond
- coordonnées géographiques des bureaux de vote parisiens :
  https://parisdata.opendatasoft.com/explore/dataset/bureaux-de-votes/information/?disjunctive.cp

Vous trouverez les indications pour installer folium et démarrer rapidement sur https://python-visualization.github.io/folium/index.html.

- **Étude de la percolation dans une grille** ⇒ *Moyen*

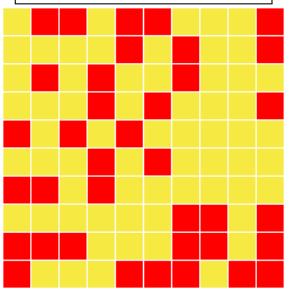
  La percolation désigne le passage d'un fluide dans une grille de sites qui peuvent être ouverts (perméables au fluide) ou fermés (imperméables au fluide).

  On modélise l'ensemble des sites par une grille de cases ouvertes ou fermées et on s'intéresse au passage d'un fluide du bord supérieur vers le bord inférieur. Lorsque le fluide remplit une case, il peut s'écouler vers l'une des 4 voisines en croix (si elle existe) : case au dessus, en dessous, à gauche ou à droite. S'il existe un chemin d'un site ouvert du bord supérieur de la grille vers un site ouvert du bord inférieur, on dit que la grille percole.

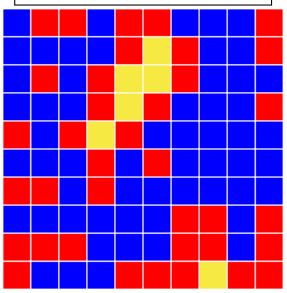
  Dans les graphiques ci-dessous, les cases jaunes sont ouvertes, les rouges sont fermées et les bleues sont remplies par le fluide.

  - **Une grille qui percole** :

Avant écoulement

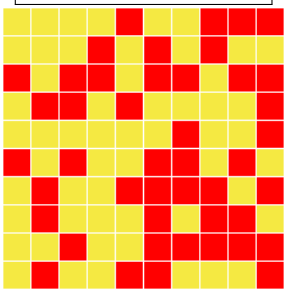


Après écoulement

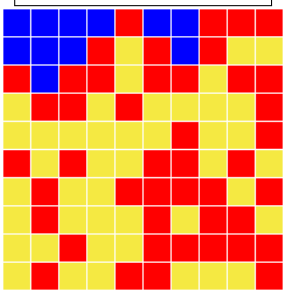


– **Une grille qui ne percole pas** :

Avant écoulement



Après écoulement



On commencera par résoudre ce problème https://e-nsi.gitlab.io/pratique/N2/666-percolation/sujet/.

Ensuite avec le module ipythonblocks dans un notebook ou pyxel dans un script Python, réaliser une petite application graphique qui simule un écoulement de fluide depuis le bord supérieure d'une grille remplie aléatoirement de sites ouverts avec une probabilité p paramétrable.

On pourra capturer l'écoulement dans un gif animé à l'aide du module imageio. Lien vers un tutoriel :

https://www.tutorialexample.com/python-create-gif-with-images-using-imageio-a-complete-

En complément on pourra étudier le seuil de percolation sur une grille $n \times n$ en mesurant pour chaque proportion p de sites ouverts, la probabilité que la grille percole à l'aide d'une méthode de Monte-Carlo : on fixe p et sur un échantillon de 100 grilles aléatoires on mesure la fréquence de grilles qui percolent.

- **Nombre de Bacon** ⇒ *Moyen*

Vous devez résoudre ce défi Codin Game à l'aide d'un parcours de graphe en largeur : https://www.codingame.com/ide/puzzle/six-degrees-of-kevin-bacon. Avec ce projet vous découvrirez un nouvelle structure de données pour modéliser des relations entre objets : les graphes et un algorithmes qui permet de déterminer le nombre minimal de sauts dans un graphe. Vous pourrez vous appuyer sur mon cours de TNSI.

- **Chiffrer comme Enigma** ⇒ *Moyen*

Vous devez résoudre ce défi Codin Game : https://www.codingame.com/training/easy/encryptiondecry
Dans ce projet vous développerez vos compétences en manipulation de chaînes de caractères et découvrirez le fonctionnement de la machine Enigma, utilisée pendant la seconde guerre mondiale par les nazis pour chiffrer leurs messages.

- **ASCII-ART** ⇒ *Moyen*

  Vous devez résoudre ce défi Codin Game : https://www.codingame.com/training/easy/ascii-art-the- Dans ce projet vous devrez créer un ASCII-ART en appliquant un algorithme. Vous développerez vos compétences en manipulation de chaînes de caractères et codage.

- **Vérification de la validité d'un numéro de carte** ⇒ *Moyen*

  L'algorithme de Luhn, ou code de Luhn, ou encore formule de Luhn est aussi connu comme l'algorithme « modulo 10 » ou « mod 10 ». C'est une formule de somme de contrôle utilisée pour valider une variété de numéros de comptes, comme les numéros de cartes bancaires, les numéros d'assurance sociale canadiens, les numéros IMEI des téléphones mobiles ainsi que pour le calcul de validité d'un numéro SIRET. Vous devez résoudre ce défi Codin Game : https://www.codingame.com/training/easy/credit-card-verifier-luhns-algorithm et proposer une interface graphique minimaliste permettant la saisie et la vérification d'un numéro avec cet algorithme.

- **Sortir d'un labyrinthe** ⇒ *Moyen*

  Vous devez résoudre ce défi Codin Game à l'aide d'un parcours de graphe en profondeur : https://www.codingame.com/ide/puzzle/maze. Avec ce projet vous découvrirez un nouvelle structure de données pour modéliser des relations entre objets : les graphes et un algorithmes qui permet d'atteindre la sortie dans un labyrinthe. Vous pourrez vous appuyer sur mon cours de TNSI.

- **Chemin le plus court dans une grille** ⇒ *Moyen*

  Vous devez résoudre ce défi Codin Game à l'aide d'un parcours de graphe en largeur : https://www.codingame.com/ide/puzzle/the-lost-child-episode-1. Avec ce projet vous découvrirez un nouvelle structure de données pour modéliser des relations entre objets : les graphes et un algorithmes qui permet de déterminer le nombre minimal de sauts dans un graphe. Vous pourrez vous appuyer sur mon cours de TNSI.

- **Convertisseur binaire/décimal** ⇒ *Moyen*

  En utilisant le module graphique nsi_ui décrit dans le manuel Hachette (pages 53 et 54) et fourni dans ressources-pf-2024.zip ou le module tkinter, créer une interface graphique de conversion d'un entier de la base dix à la base deux. L'interface devra proposer la conversion pour les entiers non signés ou signés (complément à deux).

- **Jeu de Simon** ⇒ *Moyen*

  - **Niveau 1 :** Programmer le jeu de société Simon en version texte. L'utilisateur joue contre le programme qui choisit de manière aléatoire une suite de caractères. Au début, le programme propose un caractère que le joueur doit reproduire. Ensuite, à chaque itération, un caractère supplémentaire est ajouté

    Cahier des charges :

    * Le programme choisit 10 caractères au hasard.
    * À chaque tour N :
      · le programme affiche N caractères de la suite,

        - le joueur propose une reproduction de ces N caractères,
        - En cas d'erreur, le programme s'arrête et le score du joueur est affiché.
    - Quatre niveaux de difficulté en fonction du temps laissé au joueur.
- **Niveau 2 :** Programmer le jeu de société Simon en version couleur et son. Utiliser le module pyxel.

- **Fractions égyptiennes** ⇒ *Moyen*

  Une fraction $\frac{p}{q}$ comprise entre 0 et 1 peut se décomposer en une somme de fractions de numérateur 1 et l'algorithme qui permet de construire une telle décomposition est un algorithme glouton déjà connu des Égyptiens! L'algorithme est décrit dans ce sujet d'Olympiades. Vous devez implémenter cet algorithme et proposer une petite interface graphique où l'on peut saisir une fraction pour obtenir l'affichage de sa décomposition.

- **Jeu de Morpion et Min-Max** ⇒ *Moyen à Difficile*

  Programmation d'un jeu de morpion avec ou sans interface graphique qui propose deux modes :

  1. Joueur humain 1 conte joueur humain 2 ;
  2. Joueur humain contre ordinateur qui choisit son coup avec l'algorithme Min-Max ;

  Ressource sur l'algorithme Min-Max : https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo_minmax.html.

- **Problème du voyageur de commerce** ⇒ *Difficile*

  Vous devez résoudre ce défi Codin Game à l'aide d'un algorithme glouton : https://www.codingame.com/ide/puzzle/the-travelling-salesman-problem

- **Réalisation d'une calculatrice en notation Polonaise inverse** ⇒ *Difficile*

  Écrire un programme d'émulation de calculatrice qui effectue des calculs élémentaires (addition, soustraction, multiplication, division) en notation *postfixée* ou *polonaise inverse*. Proposer une petite interface graphique en le module nsi_ui fourni dans le manuel Hachette (pages 53 et 54) ou le module tkinter. Commencer par résoudre ce problème : https://www.codingame.com/ide/puzzle/reverse-polish-nota

- **Jeu de puissance 4** ⇒ *Difficile*

  Réaliser une version du jeu de Puissance 4 pour 2 joueurs avec ou sans interface graphique. On pourra utiliser les modules pyxel ou tkinter de Python.

  Prolongement possible : un joueur joue contre l'ordinateur.

- **Site web de quizz sur le programme de première NSI** ⇒ *Difficile*

  Réaliser avec le module Flask un mini-site web permettant de saisir un thème du programme de Première NSI et générant un quizz aléatoires de questions choisies dans un fichier csv portant sur ce thème. Les scores pourraient être enregistrés dans un fichier.

  Voir ce tutoriel sur Flask :

  https://frederic-junier.gitlab.io/parc-nsi/chapitre230/php-flask-git/.

- **Sudoku** ⇒ *Difficile*

  Réaliser un solveur de Sudoku par backtracking, avec ou sans interface graphique. Source : le problème 1 de https://www.apmep.fr/IMG/pdf/CAPES_avril_2017_epreuve_info.pdf.

- **Seam carving : redimensionnement intelligent d'image** ⇒ *Difficile*

  Écrire un programme avec interface textuelle qui ouvre un fichier image au format PGM et redimensionne sa largeur en utilisant l'algorithme décrit dans le puzzle https://www.codingame.com/ide/puzzle/seam-carving. Les quatre premiers tests du puzzle sont fournis dans ressources.zip.

- **Othello et Min-Max** ⇒ *Difficile à très difficile*

  Programmation d'un jeu d'Othello /Reversi avec ou sans interface graphique qui propose deux modes :

  1. Joueur humain 1 conte joueur humain 2 ;
  2. Joueur humain contre ordinateur qui choisit son coup avec l'algorithme Min-Max ;

  Ressource sur l'algorithme Min-Max : https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo_minmax.html.

4 Quelques bibliothèques de Python qui pourraient vous être utiles

☞ Dessiner avec la tortue : turtle.

☞ Réalisation d'interfaces graphiques (suivre les liens) par ordre croissant de difficulté de prise en main, les liens pointent vers des tutoriels ou la documentation officielle :

- le module ipythonblocks permet de manipuler très facilement des grilles de blocs colorés dans un notebook jupyter comme ceux utilisés sur le service Capytale de l'ENT ;
- le module pyxel.
- le module nsi_ui fourni dans le manuel Hachette (pages 53 et 54) et dans .
- le module tkinter

☞ Traitement d'images : **PIL** ou son fork **Pillow** : https://pypi.python.org/pypi/Pillow/

☞ Réalisation de graphiques pour les mathématiques ou la physique (possibilité d'animation) : **matplotlib** : http://matplotlib.org/

☞ Traitement avancé de données : https://pandas.pydata.org/

☞ Collecte de données sur le Web :

- récupérer les données avec **requests** : https://realpython.com/python-requests/
- « parser » du HTML avec **beautiful soup** : https://realpython.com/beautiful-soup-web-scraper-python/.

5 Quelques outils pour la gestion de projet ou le travail collaboratif

- ☞ Cours et ressources en ligne sur https://frederic-junier.gitlab.io/parc-nsi/projets/.
- ☞ Partage de fichiers, pad, articles de blog, notebooks Python dans l'ENT.
- ☞ Partage de scripts Python et test en ligne : https://console.basthon.fr/. ou le service Capytale de l'ENT.
- ☞ Documents textuels collaboratifs (avec possibilité de chat) : **Framapad** accessible depuis https://framapad.org/.
- ☞ Feuilles de calculs collaboratives : **Framacalc** accessible depuis https://framacalc.org/.
- ☞ Réalisation de carte mentale : **Framindmap** accessible depuis https://framindmap.org/c/maps/.