

## Table des matières

<b>1 Introduction</b>	<b>1</b>	2.2 Génération de versions multiples . . . . .	3
1.1 A quoi sert ce logiciel . . . . .	1	<b>3 Utilisation personnalisée</b>	<b>3</b>
1.2 Architecture . . . . .	1	3.1 Dossier inputs . . . . .	3
1.2.1 Modules principaux . . . . .	2	3.1.1 Structure des fichiers .sty . . . . .	3
1.3 Dépendances externes . . . . .	2	3.1.2 Syntaxe des variables . . . . .	4
<b>2 Fonctionnalités</b>	<b>2</b>	3.2 Dossier json-productions . . . . .	5
2.1 Énoncé abstrait . . . . .	2		

## 1. Introduction

### 1.1 A quoi sert ce logiciel

Ce logiciel est un **gestionnaire** d'énoncés destinés à être utilisés comme **automatismes**.

Il pourra toutefois être utilisé à d'autres fins par modification de la configuration.

Il permet, avec une syntaxe légère, de construire des énoncés respectant les contraintes suivantes :

1. Les énoncés doivent être écrits 100% en langage  $\text{\LaTeX}$ .
2. Définir des **variables** au début d'énoncé, et répond à la question en utilisant **exclusivement ces variables**.
3. Organiser et répertorier les énoncés produits.
4. Permettre de générer des versions multiples grâce à des générateurs.
5. Permettre une modification manuelle **facilitée** par modification des variables.

### 1.2 Architecture

Le projet s'organise selon l'arborescence suivante :

```

implementer_enonce/
├── main.py
├── enonce_implementator.py
├── inputs/
│   ├── 6ème/
│   ├── 5ème/
│   ├── 4ème/
│   ├── 3ème/
│   ├── 2nde/
│   ├── 1ere/
│   │   └── [Thème]/
│   │       └── exercice.sty
├── json_productions/
│   └── [même structure que inputs/]
├── productions/
├── modules/
│   ├── UI_v2.py
│   └── version_maker.py

```

```

├─ numbers_def.py
├─ exo_number.py
├─ number_generators_manager.py
├─ number_generators/
│   ├── arithmetic_number_def.py
│   ├── float_number_def.py
│   └─ number_generator.py
├─ abstractor/
│   ├── abstract_exo_ui.py
│   └─ question_abstractor.py
└─ prompts/

```

### 1.2.1 Modules principaux

#### UI\_v2.py

Interface graphique avec **CustomTkinter**. Classe `ExerciseEditor` gérant l'édition multi-onglets et l'intégration IA.

#### version\_maker.py

Parsing du format %% et génération des versions. Gère les types SHORT, HELP, FULL, QCM.

#### numbers\_def.py

Analyse syntaxique des définitions de variables et génération des valeurs aléatoires selon contraintes.

#### number\_generators\_manager.py

Chargement dynamique des générateurs depuis `modules/number_generators/`. Système extensible.

## 1.3 Dépendances externes

Le module graphique est dépendant d'un module de **QFGen**.

Ainsi il faut que les deux programmes soient dans le même dossier parent.

Dossier parent

```

├─ implementer_enonce/
└─ QFGen/

```

## 2. Fonctionnalités

### 2.1 Énoncé abstrait

Par appui sur le bouton « Abstraire un énoncé », une modale de texte s'ouvre.

Cette modale est en réalité destinée à recevoir un prompt qui peut être de plusieurs nature :

1. Un exercice venu de votre collection.
2. Un prompt basique.
3. Un exercice partiel ( par exemple récupéré par extraction de texte ) associé à un prompt.

4. Dans tous les cas, plus il y aura de détails, plus l'agent réalisera votre idée.

De la même manière, Plus le code  $\LaTeX$  est de qualité, moins l'agent aura de travail de production.

Un agent s'occupe alors de générer les éléments suivants :

#### Variables

Une syntaxe particulière est réservée aux variables pour définir les bornes de la génération, la nature du nombre généré, valeurs interdites, liste de choix...

#### Code $\LaTeX$

L'agent génère en fait un code  $\LaTeX$  respectant les contraintes de segmentation permettant à l'application d'interpréter l'énoncé, la solution détaillée, et la solution courte.

#### Métadonnées

Quelques métadonnées associées, notamment le thème de l'énoncé.

## 2.2 Génération de versions multiples

Une fois un énoncé saisi ou généré, il est possible de l'enregistrer en sélectionnant les différents paramètres dans le menu.

Ces paramètres sont simplement les **noms des dossiers** dans la **base de données** alimentée par l'application. Cela permet d'**organiser** les fichiers produits.

Ils pourront donc être utilisés par d'autres applications, par exemple **QFGen**.

Pour réaliser mes **automatismes**, je génère systématiquement 30 versions pour chaque énoncé.

## 3. Utilisation personnalisée

Les productions sont de deux natures :

### Inputs

Des fichiers « .tex » contenant le code abstrait.  
Ces codes ne sont pas directement compilables, ce sont les **modèles** servant à la génération.

### json\_productions

Les versions générées sont stockées dans un dictionnaire **.json**. Un simple algorithme python permet de sélectionner n'importe quelle version et de manipuler aisément son énoncé et tous les éléments qui l'accompagnent.

### 3.1 Dossier inputs

Il est possible de directement produire les fichiers d'énoncés, en les stockant dans le dossier « inputs ». Cela permet l'intervention de logiciels externes pour les productions d'énoncé qui respectent ce format ( voir les exemples déjà générés. )

L'application les détectera automatiquement et il sera possible de générer les versions multiples.

#### 3.1.1 Structure des fichiers .sty

Les fichiers .sty suivent une structure **stricte** avec des séparateurs **%%**. L'ordre des compartiments est **impératif** :

1. **Nombre de versions** : Un entier seul sur la première ligne
2. **%%**
3. **Variables** : Définitions avec syntaxe spéciale (voir section suivante)
4. **%%**
5. **Énoncé** : Code  $\text{\LaTeX}$  de la question
6. **%%**
7. **Solution détaillée** : Résolution complète avec explications
8. **%%**
9. **Versión QCM** : Liste `enumerate` avec la bonne réponse en dernier item
10. **%%**
11. **Réponse courte** : Code concaténé sur une seule ligne
12. **%%**
13. **Thème** : Nom du thème (ex : « Polynomes degré 2 »)
14. **%%**

### Contraintes critiques :

- **Réponse courte (compartiment 11) : AUCUN commentaire  $\LaTeX$  (pas de %).** Le code doit être **entièrement concaténé** sur une ligne unique sans retours chariots.
- **Qualité du code :** L'application ne dispose **pas de système de prévisualisation**. Une **rigueur maximale** est exigée car les erreurs  $\LaTeX$  ne seront détectées qu'à la compilation finale.
- **Respect de l'ordre :** Les compartiments doivent apparaître dans l'ordre exact indiqué. Toute inversion provoquera des erreurs de parsing.

### 3.1.2 Syntaxe des variables

Le système supporte plusieurs syntaxes pour la définition de nombres aléatoires :

#### Syntaxe de base (ExoNumber)

```
\def\n{2<=x<20 int}           % Entier entre 2 et 19
\def\d{1<=x<=5 float decimalesD2} % Float avec 2 décimales
\def\frac{1<x<=10 x!=0 frac}    % Fraction
\def\nb{4<x x!=6 int decimalesG3} % Int à 3 chiffres
\def\list[option1,option2,...]  % Choix aléatoire
```

#### Paramètres disponibles

Bornes	Exclusions
$a \leq x \leq b$ , $a < x < b$ , $a \leq x < b$ , $a < x \leq b$ . Si borne absente : min=-100, max=100.	$x \neq \text{valeur}$ pour exclusion des valeurs (multiples possibles : $x \neq 2$ $x \neq 5$ ).
Type	Décimales droite
int (entier), float (décimal), frac (fraction $\dfrac{\text{num}}{\text{den}}$ ).	decimalesDn pour n décimales après la virgule (float uniquement).
Décimales gauche	Listes
decimalesGn pour n chiffres dans la partie entière.	$\def\var[val1,val2,...]$ choisit aléatoirement parmi les options.

#### Générateurs personnalisés :

Le système charge dynamiquement les générateurs depuis modules/number\_generators/.

Exemple d'utilisation :

```
ArithmeticNumber(
  allowed_generators=[2, 3, 5, 7, 11, 13],
  prime_length=[2, 4],
  inf=100,
  sup=1000,
  name="n"
)
```

- **allowed\_generators :** Liste des facteurs premiers autorisés
- **prime\_length :** Nombre de facteurs (entier ou intervalle [min, max])
- **inf/sup :** Bornes du résultat final
- **name :** Nom de la variable ("auto" pour auto-génération)

**Extensibilité :** Tout nouveau générateur héritant de NumberGenerator placé dans modules/number\_generators/ sera automatiquement disponible.

## 3.2 Dossier json-productions

Ce dossier est destiné à être alimenté par l'application seulement.  
Tout ce qu'il contient est susceptible d'être écrasé par des manipulations futures.  
Il respecte la même architecture que le dossier « inputs »