Cycle : lycée qualifiant
Filière : Mathématiques

Projet de fin de formation

Python et théorie des nombres

Réalisé par :
Majjaj Mohamed

Sous la direction du :
Pr. DERFOUFI Younes

Remerciements

Je tiens à exprimer ma profonde gratitude à mon professeur encadrant, Monsieur **Younes Derfoufi**, pour son accompagnement, ses précieux conseils et sa disponibilité tout au long de ce projet. Son expertise et ses encouragements constants ont été déterminants dans la réussite de ce travail.

Je remercie également Monsieur **Mohammed Salhi** pour son soutien pédagogique et son suivi attentif durant ma formation.

Mes remerciements s'adressent aussi à l'ensemble des membres du jury pour l'intérêt qu'ils ont porté à ce travail, leurs remarques constructives et leur bienveillance.

Enfin, je remercie tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce projet.

Résumé du projet

Ce projet de fin de formation vise à moderniser l'enseignement de la théorie des nombres au lycée qualifiant, en s'appuyant sur la programmation avec **Python**. À travers une approche pédagogique interactive, il explore les notions fondamentales telles que la parité, la divisibilité, la factorisation en nombres premiers, le calcul du PGCD/PPCM, les tests de primalité (simples et probabilistes), ainsi que les nombres premiers de **Mersenne**.

L'objectif principal est d'aider les élèves à mieux comprendre ces concepts abstraits grâce à des scripts Python simples, visuels et adaptés. Le projet inclut aussi le développement d'une application avec **PyQt5**, offrant une interface graphique intuitive qui permet aux utilisateurs de tester des nombres, de calculer le PGCD/PPCM, de factoriser, et de générer des nombres de Mersenne sans écrire une seule ligne de code.

Ce projet s'adresse principalement aux enseignants stagiaires et aux élèves passionnés de mathématiques et de programmation. Il montre comment Python peut être un outil pédagogique puissant et accessible pour enrichir les apprentissages en mathématiques.

Table des matières

INTRODUCTION GÉNÉRALE ET L'OBJECTIFS DU PROJET

1.1 Contexte général du projet

Dans le cadre de la formation des enseignants stagiaires du secondaire, l'usage des outils numériques s'impose comme un enjeu central pour l'enseignement des mathématiques. En effet, les environnements interactifs permettent non seulement de renforcer l'engagement des élèves, mais aussi de rendre plus concrets des concepts abstraits grâce à des visualisations et des simulations accessibles.

Parmi ces outils, la programmation avec **Python** offre de nombreuses possibilités pour illustrer des notions mathématiques complexes de manière claire et interactive. La théorie des nombres, domaine riche mais souvent perçu comme difficile, peut ainsi être explorée à travers des scripts simples, des algorithmes de calcul, et des interfaces conviviales.

Ce projet s'inscrit dans une démarche de modernisation de l'enseignement, en mettant l'accent sur l'expérimentation, l'interactivité et la construction progressive des savoirs, tout en développant chez les futurs enseignants

des compétences en algorithmique et en conception d'outils pédagogiques.

1.2 Problématique

L'enseignement de la théorie des nombres, notamment des notions comme la primalité, la divisibilité ou encore les tests probabilistes, demeure parfois abstrait et difficile à visualiser pour les élèves.

Comment faire en sorte que ces concepts deviennent intuitifs, motivants et concrets ?

En parallèle, l'introduction de Python dans les programmes de mathématiques permet d'illustrer les notions de façon dynamique. Il devient alors pertinent de se demander :

Comment Python peut-il faciliter la compréhension et l'exploration des notions liées aux nombres premiers, à la factorisation et aux tests de primalité ?

1.3 Objectifs du projet

1.3.1 Objectif principal

Développer une approche pédagogique interactive, à l'aide de Python, pour l'étude des nombres premiers, de leur factorisation et des tests de primalité, avec une ouverture vers les nombres premiers de Mersenne.

1.3.2 Objectifs spécifiques

- Présenter les notions fondamentales de la théorie des nombres de manière claire.
- Utiliser Python pour illustrer :
  - La détection des nombres premiers ;
  - Les algorithmes de factorisation ;
  - Les tests de primalité (naïfs et probabilistes).
- Créer de petits scripts Python simples et accessibles aux élèves.
- Introduire les nombres de Mersenne et leur lien avec la primalité.
- Favoriser une pratique active des mathématiques par l'expérimentation algorithmique.

1.4 Présentation générale de l'application

L'application développée dans ce projet repose principalement sur le langage Python, accompagné de bibliothèques comme :

— **sympy** : pour la manipulation symbolique et les outils de théorie des nombres ;

— **math** : pour les fonctions mathématiques de base ;

— **random** : pour certains tests probabilistes ;

— **PyQt5** (optionnel) : pour la réalisation d'une interface simple et conviviale.

L'élève pourra ainsi :

— Vérifier si un nombre est premier ;

— Factoriser un nombre entier ;

— Appliquer des tests de primalité sur de grands nombres ;

— Générer et analyser des nombres de Mersenne.

1.5 Public cible

— Professeurs stagiaires de mathématiques souhaitant enrichir leurs pratiques pédagogiques.

— Élèves du lycée qualifiant, intéressés par les mathématiques, la programmation ou la cryptographie.

— Clubs scientifiques ou ateliers de programmation en milieu scolaire.

1.6 Méthodologie adoptée

Le travail a suivi les étapes suivantes :

1. Analyse des notions fondamentales de la théorie des nombres (nombres premiers, divisibilité, etc.).
2. Conception d'algorithmes simples et de fonctions Python adaptées au niveau des élèves.
3. Implémentation des scripts avec des explications claires et illustrées.

4. Test et validation pédagogique auprès de quelques apprenants ou enseignants.
5. Développement d'une interface graphique avec PyQt5.

QUELQUES NOTIONS SUR LA THÉORIE DES NOMBRES

2.1 Introduction à la théorie des nombres

La théorie des nombres est une branche des mathématiques pures qui étudie les propriétés des entiers naturels. Elle comprend l'étude des nombres premiers, des congruences, de la divisibilité, et des fonctions arithmétiques. Dans ce projet, nous nous intéressons plus particulièrement aux nombres premiers, à leur décomposition, aux tests de primalité, ainsi qu'aux nombres de Mersenne.

2.1.1 Définition : Nombres pairs et impairs

Soit $\mathbb{N}$, l'ensemble des entiers naturels.
Un entier n est pair s'il existe un entier k tel que :

$$n = 2k$$

Un entier n est impair s'il existe un entier k tel que :

$$n = 2k + 1$$

2.1.2 Code Python : Vérification de parité

```
from sympy import Mod


def test_parite(n):
    if Mod(n, 2) == 0:
        print('Pair')
    else:
        print('Impair')


n = int(input("n = "))
test_parite(n)
```

Explication : `Mod(n, 2)` retourne le reste de la division de n par 2. Si le reste est 0, le nombre est pair. Sinon, il est impair.

2.2 Critères de divisibilité

Un nombre naturel est divisible par :

- **2** : si le chiffre des unités est pair.
- **3** : si la somme des chiffres est divisible par 3.
- **4** : si le nombre formé par les deux derniers chiffres est divisible par 4.
- **5** : si le chiffre des unités est 0 ou 5.
- **8** : si le nombre formé par les trois derniers chiffres est divisible par 8.
- **9** : si la somme des chiffres est divisible par 9.

2.2.1 Code Python : Critères de divisibilité

```
from sympy import Mod


def cretere_de_devisible(n):
    if Mod(n, 2) == 0:
        print(f"{n} est divisible par 2")
    if Mod(sum([int(d) for d in str(n)]), 3) == 0:
        print(f"{n} est divisible par 3")
    if Mod(int(str(n)[-2:]), 4) == 0:
        print(f"{n} est divisible par 4")
    if str(n)[-1] in ['0', '5']:
        print(f"{n} est divisible par 5")

    if Mod(int(str(n)[-3:]), 8) == 0:
        print(f"{n} est divisible par 8")

    if Mod(sum([int(d) for d in str(n)]), 9) == 0:
        print(f"{n} est divisible par 9")


n = int(input("n = "))
cretere_de_divisible(n)
```

2.3 Nombres premiers

2.3.1 Définition

Un entier naturel n est dit premier s'il a exactement deux diviseurs positifs : 1 et n.

Exemples :

$2, 3, 5, 7, 11, 13, 17$ sont des nombres premiers.

Par contre, $4, 6, 8, 9$ ne le sont pas car ils ont d'autres diviseurs.

Remarques :

— 2 est le seul nombre premier pair.
— Tous les autres nombres premiers sont impairs.
— Il y a une infinité de nombres premiers (preuve par contradiction d'Euclide).

Liste des premiers nombres premiers :

$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, \ldots$

2.4 Test de primalité simple

Pour étudier la primalité d'un nombre entier naturel n, on cherche tous les nombres premiers p tels que $p \leq \sqrt{n}$.

Si n est divisible par l'un de ces nombres alors n n'est pas un nombre premier, sinon n est premier.

Exemple :

Prenons $n = 29$.

— Les nombres premiers $\leq \sqrt{29}$ sont $2, 3, 5$.
— $29 \div 2 = 14.5$ (non divisible)
— $29 \div 3 \approx 9.67$ (non divisible)
— $29 \div 5 = 5.8$ (non divisible)

Aucun de ces diviseurs ne divise29. Donc, 29 est un nombre premier.

Code Python pour le test de primalité avec `sympy` :

```
from sympy import isprime

# Fonction pour tester la primalite
def est_premier_sympy(n):
    return isprime(n)

# Exemple d'utilisation
print(est_premier_sympy(29))  # True
print(est_premier_sympy(30))  # False
```

2.5 Test de primalité avancée :

Le **test de primalité de Miller-Rabin** est un test probabiliste permettant de déterminer si un nombre entier n est probablement premier.

Principe :

Soit n un nombre impair que l'on écrit sous la forme :

$$n = 2^s \times d + 1$$

avec$d$ impair et $s \geq 1$. On choisit un entier a tel que $1 \leq a \leq n-1$.

Le test de Miller-Rabin repose sur les deux conditions suivantes :

1. $a^d \equiv 1 \mod n$

2. $a^{2^i \times d} \equiv -1 \mod n$ pour un certain $i \in \{0, 1, \ldots, s-1\}$.

– Si **l'une des deux conditions est vérifiée**, on dit que n passe le test de Miller-Rabin pour la base a.

– Si **aucune des deux conditions n'est vérifiée**, alors n est composé.

Un nombre premier passe le test pour tout a.

Un nombre composé peut passer le test pour certains a, mais avec une probabilité de $\leq \frac{1}{4}$.

Exemple

Prenons $n = 561$, un nombre connu pour être un **nombre de Carmichael**, c'est-à-dire qu'il passe certains tests de primalité bien qu'il soit composé.

— Écrivons $561 - 1 = 560$ sous la forme $2^s \times d$.

$$560 = 2^4 \times 35$$

— Prenons $a = 2$.

1. Calcul de $a^d \mod n$:

$$2^{35} \mod 561 = 263 \neq 1 \mod 561$$

2. Vérifions la seconde condition pour $i = 0, 1, 2, 3$:

$$2^{2^0 \times 35} \mod 561 = 263$$

$$2^{2^1 \times 35} \mod 561 = 166$$
$$2^{2^2 \times 35} \mod 561 = 67$$
$$2^{2^3 \times 35} \mod 561 = 1$$

Aucune des conditions n'est satisfaite, donc 561 est composé.

Le test de Miller-Rabin peut être implémenté facilement en utilisant la bibliothèque sympy de Python.

```python
from sympy import isprime

def test_miller_rabin(n):
    """
    Teste la primalite d'un nombre entier 'n' en
       utilisant la fonction isprime() de SymPy.
    """
    if isprime(n):
        print(f"{n} est probablement premier.")
    else:
        print(f"{n} est compose.")

# Exemple d'utilisation
test_numbers = [561, 1105, 1729, 19, 29, 97]
for num in test_numbers:
    test_miller_rabin(num)
```

2.6 Décomposition en facteurs premiers

2.6.1 Définition

Soit $a \in \mathbb{N}^* \setminus \{1\}$, alors,$a$ s'écrit sous la forme d'un produit de plusieurs facteurs premiers, que l'on appelle la décomposition en facteurs premiers du nombre a.

Exemple : $360 = 2^3 \times 3^2 \times 5$

2.6.2 Thérème

Soit $a \in \mathbb{N}^* \setminus \{1\}$. Il existe des entiers naturels non nuls $\alpha_1, \alpha_2, \ldots, \alpha_n$ et des nombres premiers distincts deux à deux $p_1, p_2, \ldots, p_n$ tels que la décomposition unique de a s'écrit sous la forme :

$$a = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \cdots \times p_n^{\alpha_n}.$$

Exemple :
Prenons $a = 84$.
Sa décomposition en facteurs premiers est :

$$84 = 2^2 \times 3^1 \times 7^1,$$

où $p_1 = 2$, $p_2 = 3$, $p_3 = 7$ sont des nombres premiers distincts, et les exposants sont $\alpha_1 = 2$, $\alpha_2 = 1$, $\alpha_3 = 1$.

Propriété (fondamentale)

Tout nombre entier naturel $n > 1$ admet une décomposition unique en produit de nombres premiers (à l'ordre près). C'est le théorème fondamental de l'arithmétique.

2.6.3 Code Python : Décomposition en facteurs premiers

```
from sympy import factorint

def decomposition_facteurs_premiers(n):
    return factorint(n)
```

2.7 PGCD (Plus Grand Commun Diviseur)

2.7.1 Définition

Soient a et b deux entiers naturels non nuls.

Le plus grand commun diviseur de a et b, noté $\gcd(a, b)$ ou $a \wedge b$, est le plus grand entier naturel qui divise à la fois a et b.

2.7.2 Théorème

Soient a et b deux entiers naturels tels que $a, b \geq 2$.

Le plus grand commun diviseur de a et b, noté $\gcd(a, b)$, est le produit des facteurs premiers communs à a et b, chacun élevé à la puissance du plus petit exposant trouvé dans la décomposition en facteurs premiers de a et b.

$$\gcd(a, b) = \prod_{i=1}^{n} p_i^{\min(\alpha_i, \beta_i)}$$

où :

– p_i sont les facteurs premiers communs à a et b. – α_i et β_i sont les exposants correspondants dans les décompositions de a et b.

Exemple :

Prenons $a = 84$ et $b = 36$.

La décomposition en facteurs premiers est :

$$84 = 2^2 \times 3 \times 7$$

$$36 = 2^2 \times 3^2$$

Les facteurs premiers communs sont 2^2 et 3.

Donc, le PGCD est :

$$\gcd(84, 36) = 2^2 \times 3 = 12$$

On peut aussi écrire :

$$84 \wedge 36 = 12$$

2.8 Entiers premiers entre eux

2.8.1 Définition

Deux entiers a et b sont dits **premiers entre eux** (ou **étrangers**) si leur plus grand commun diviseur est égal à 1.

En d'autres termes :

$$\gcd(a, b) = 1 \quad \text{ou} \quad a \wedge b = 1$$

Exemple :

Prenons $a = 8$ et $b = 15$.

Les décompositions en facteurs premiers sont :

$$8 = 2^3$$

$$15 = 3 \times 5$$

Les ensembles de facteurs premiers sont distincts, donc ils n'ont aucun facteur premier commun.

Ainsi :

$$\gcd(8, 15) = 1$$

On peut donc conclure que 8 et 15 sont premiers entre eux.

2.8.2 Code Python : Calcul du PGCD

```python
from sympy import gcd

# Fonction pour calculer le PGCD de deux entiers
def pgcd(a, b):
    return gcd(a, b)

# Exemple d'utilisation generale
x = int(input("Entrez le premier nombre : "))
y = int(input("Entrez le deuxieme nombre : "))
print(f"Le PGCD de {x} et {y} est : {pgcd(x, y)}")
```

2.9 PPCM (Plus Petit Commun Multiple)

2.9.1 Définition

Soient a et b deux entiers naturels non nuls. Le PPCM, noté $\text{lcm}(a, b)$, est le plus petit entier divisible par a et b.

2.9.2 Théorème

Le plus petit commun multiple de a et b, noté $\text{ppcm}(a, b)$, est le produit de tous les facteurs premiers communs et non communs de a et b, chacun élevé à la puissance du plus grand des exposants trouvés dans leurs décompositions en facteurs premiers.

En d'autres termes :

$$\text{ppcm}(a, b) = \prod_{i=1}^{n} p_i^{\max(\alpha_i, \beta_i)}$$

où :

– p_i sont les facteurs premiers communs ou non communs à a et b. – α_i et β_i sont les exposants correspondants dans les décompositions de a et b.

Exemple :

Prenons $a = 12$ et $b = 18$.

Les décompositions en facteurs premiers sont :

$$12 = 2^2 \times 3^1$$

$$18 = 2^1 \times 3^2$$

Les facteurs premiers sont 2 et 3.

Les exposants maximaux sont :

– Pour 2 : $\max(2, 1) = 2$ – Pour 3 : $\max(1, 2) = 2$

Ainsi, le PPCM est :

$$\text{ppcm}(12, 18) = 2^2 \times 3^2 = 4 \times 9 = 36$$

Remarques

— $\gcd(a, b) = \gcd(b, a)$

— $\gcd(1, a) = 1$

— $\gcd(a, a) = a$

— $\text{ppcm}(a, b) = \text{ppcm}(b, a)$

— $\text{ppcm}(1, a) = a$

— $\text{ppcm}(a, a) = a$

— Relation fondamentale :

$$\gcd(a, b) \times \text{ppcm}(a, b) = a \times b$$

2.9.3 Code Python : Calcul du PPCM

```python
from sympy import lcm

# Fonction pour calculer le PPCM de deux entiers
def ppcm(a, b):
    return lcm(a, b)

# Exemple d'utilisation generale
x = int(input("Entrez le premier nombre : "))
y = int(input("Entrez le deuxieme nombre : "))
print(f"Le PPCM de {x} et {y} est : {ppcm(x, y)}")
```

2.10 Nombres premiers de Mersenne

2.10.1 Définition :

Un nombre de Mersenne est un nombre de la forme : $M_n = 2^n - 1$, où n est un entier naturel.

Exemples :

- Si $n = 2$: $M_2 = 3 \rightarrow$ premier
- Si $n = 3$: $M_3 = 7 \rightarrow$ premier
- Si $n = 4$: $M_4 = 15 \rightarrow$ non premier

Remarques :

- Si M_n est premier, alors n doit être premier.
- (mais l'inverse n'est pas toujours vrai)
- Les plus grands nombres premiers connus sont souvent des nombres de Mersenne.

2.10.2 Code Python pour tester les nombres premiers de Mersenne :

```python
from sympy import isprime


def mersenne_premiers(max_exposant):
    for n in range(2, max_exposant + 1):
        m = 2**n - 1
        if isprime(m):
            print(f"M{n} = {m} est premier")


# Appel de la fonction
mersenne_premiers(20)
```

APPLICATION SUR LA THÉORIE DES NOMBRES

3.1 Motivation et introduction

La théorie des nombres est une branche fondamentale des mathématiques qui étudie les propriétés des entiers. Elle est à la fois ancienne et toujours en pleine expansion, trouvant aujourd'hui des applications concrètes en cryptographie, en sécurité informatique, en algorithmique, et même en intelligence artificielle. Toutefois, son abstraction peut rendre son apprentissage ardu, surtout lorsqu'elle est abordée uniquement de manière théorique.

Dans ce contexte, l'utilisation du langage `Python` s'avère particulièrement pertinente. Python permet de mettre en œuvre de manière simple et efficace des algorithmes mathématiques, tout en offrant une syntaxe claire, accessible aux étudiants comme aux chercheurs. En couplant Python à une interface graphique, comme celle offerte par la bibliothèque `PyQt5`, il devient possible de rendre la théorie des nombres plus interactive, plus visuelle, et donc plus engageante.

L'objectif principal de cette application est de fournir un outil pédagogique et expérimental pour explorer certains aspects fondamentaux de la

théorie des nombres, tels que la primalité, la factorisation, les nombres de Mersenne, ou encore les tests probabilistes. Grâce à l'interface graphique conçue avec `PyQt5`, l'utilisateur peut interagir avec les algorithmes sans avoir à manipuler du code, ce qui favorise une meilleure compréhension des concepts sous-jacents.

Ce projet illustre donc comment la programmation et l'interaction homme-machine peuvent enrichir l'apprentissage et l'exploration de domaines mathématiques classiques, en les rendant à la fois plus accessibles et plus ludiques.

3.2 Présentation de l'interface graphique

3.2.1 Introduction à PyQt5

Pour réaliser cette application, nous avons utilisé `PyQt5`, une bibliothèque Python permettant de concevoir des interfaces graphiques complètes, multiplateformes et réactives.
Elle fournit des composants prêts à l'emploi tels que des fenêtres, boutons, champs de texte, etc., que nous avons intégrés pour permettre une interaction fluide avec les différentes fonctionnalités du projet.

3.2.2 Fonctionnalités principales de PyQt5

- **Widgets variés** : boutons, zones de texte, cases à cocher, labels, etc.
- **Gestion des événements** : clics sur les boutons, entrées clavier, etc.
- **Fenêtres et dialogues** : pour créer des interfaces simples ou complexes.
- **Personnalisation graphique** : choix des couleurs, des polices, de la disposition des éléments.
- **Multiplateforme** : fonctionne sur Windows, macOS et Linux sans changer le code.

3.2.3 Architecture de PyQt5

— **QtCore** : pour les bases du programme (horloges, fichiers, signaux/slots, etc.).

— **QtGui** : pour les images, polices et couleurs.

— **QtWidgets** : contient les éléments d'interface comme les boutons et les fenêtres.

— **QtMultimedia**, **QtWebEngine** : pour l'audio, la vidéo ou le web (optionnels).

3.2.4 Structure d'une application PyQt5

Une application `PyQt5` se construit généralement en quatre étapes :

1. Créer l'objet principal de l'application (`QApplication`).
2. Définir la fenêtre et les éléments (widgets) qu'elle contient.
3. Connecter les événements (par exemple : clic sur un bouton).
4. Lancer l'application avec une boucle d'événements (`app.exec_()`).

3.2.5 Exemple simple dune fenêtre

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget,
   QPushButton


class App(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        btn = QPushButton('Cliquez-moi', self)
        btn.clicked.connect(self.on_click)
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        self.setWindowTitle('Application PyQt5')
        self.setGeometry(100, 100, 300, 200)

    def on_click(self):
        print("Le bouton a ete clique !")


if __name__ == "__main__":
    app = QApplication(sys.argv)
    fenetre = App()
    fenetre.show()
    sys.exit(app.exec_())
```

3.2.6 Conclusion

En résumé, `PyQt5` est un excellent choix pour développer des interfaces graphiques modernes en Python. Il offre une grande flexibilité, une large gamme de widgets, ainsi qu'une forte intégration multiplateforme. Grâce à ses nombreuses fonctionnalités, il permet de concevoir des applications à la fois puissantes et conviviales, adaptées à des contextes pédagogiques comme professionnels.

3.3 Description de l'interface graphique

Dans ce section on va faire une description de l'interface graphique, qu'on a fabriquée avec la bibliothèque PyQt5 : En commençant d'abord par l'interface principale, comme le montre la figure suivante :

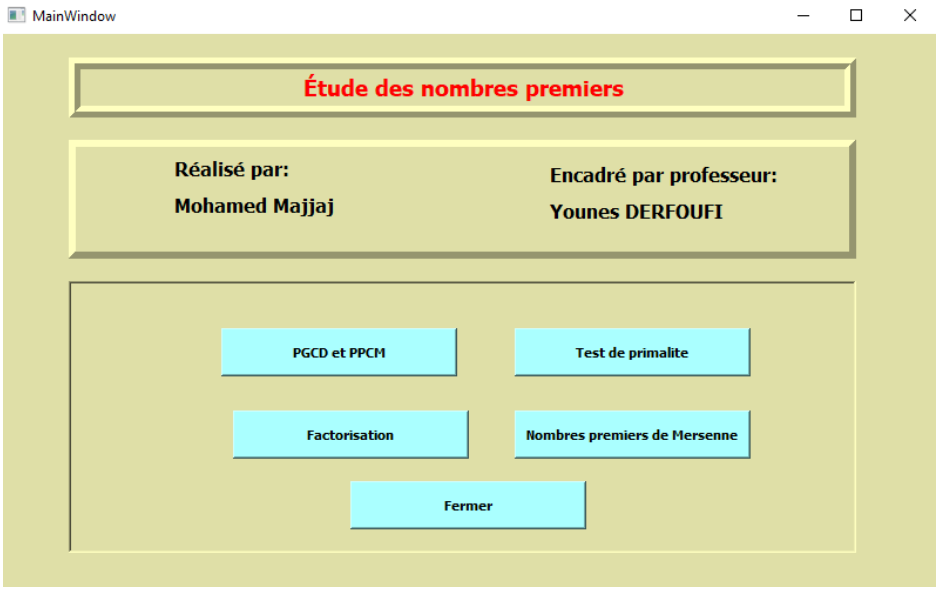


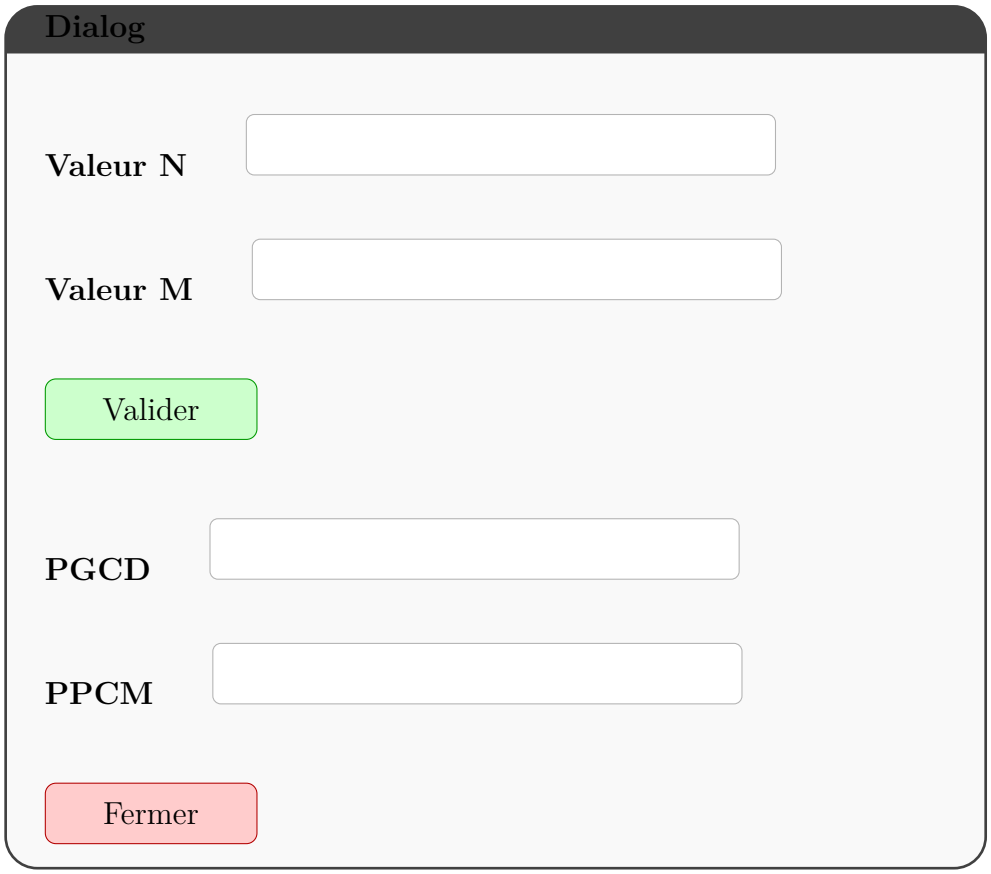
FIGURE 3.1 – L'interface principale de l'application

Dans cette interface, on a les labels, les frames, ,mais on s'interessant seulement à l'étude des 5 boutons suivants :

— **Bouton 1 :** calculer pgcd et ppcm
— **Bouton 2 :**Effectuer un test de primalité
— **Bouton 3 :**Factoriser un entier
— **Bouton 4 :**Générer les nombres premiers de Mersenne
— **Bouton 5 :**Quitter l'application

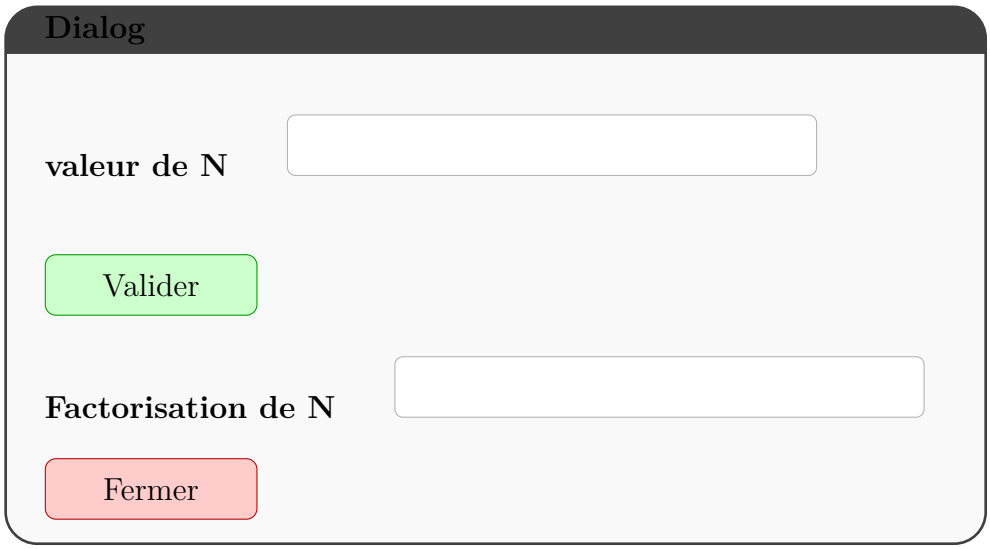
Chaque bouton ouvre une fenêtre dédiée avec des champs de saisie et des résultats dynamiques.

3.3.1 Bouton 1 : calcul pgcd et ppcm



Ce bouton ouvre une fenêtre permettant à l'utilisateur de saisir un entier N et M et en clique sur le bouton validr Le programme calcule automatiquement le pgcd et ppcm `sympy.isprime()`.

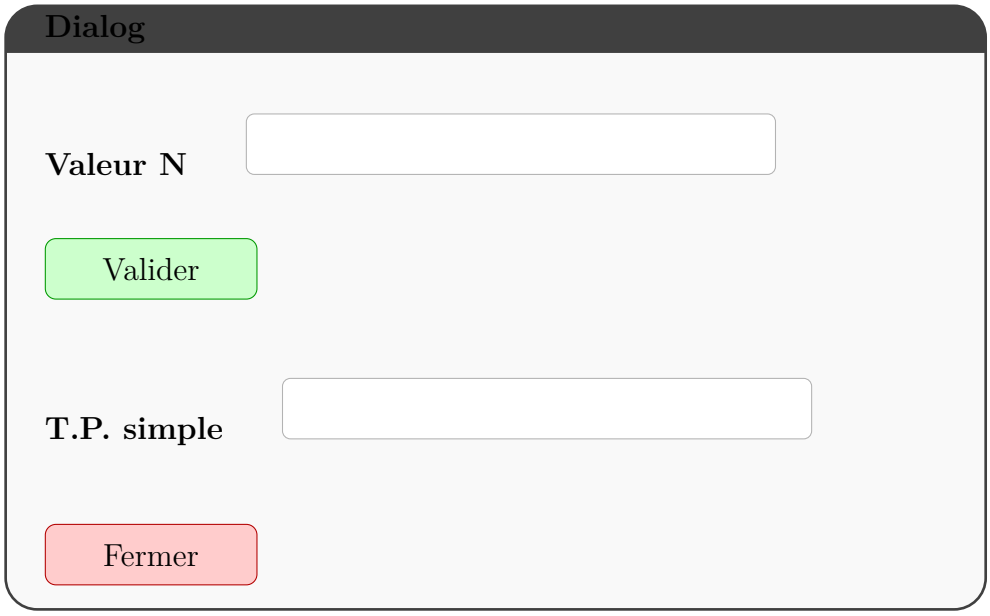
3.3.2 Bouton 2 :Factorisation



Cette fonctionnalité utilise `sympy.factorint(n)` pour afficher la factorisation d'un entier.
Exemple :

$$\text{Entrée : } 90 \quad \longrightarrow \quad \text{Sortie : } 2 \times 3^2 \times 5$$

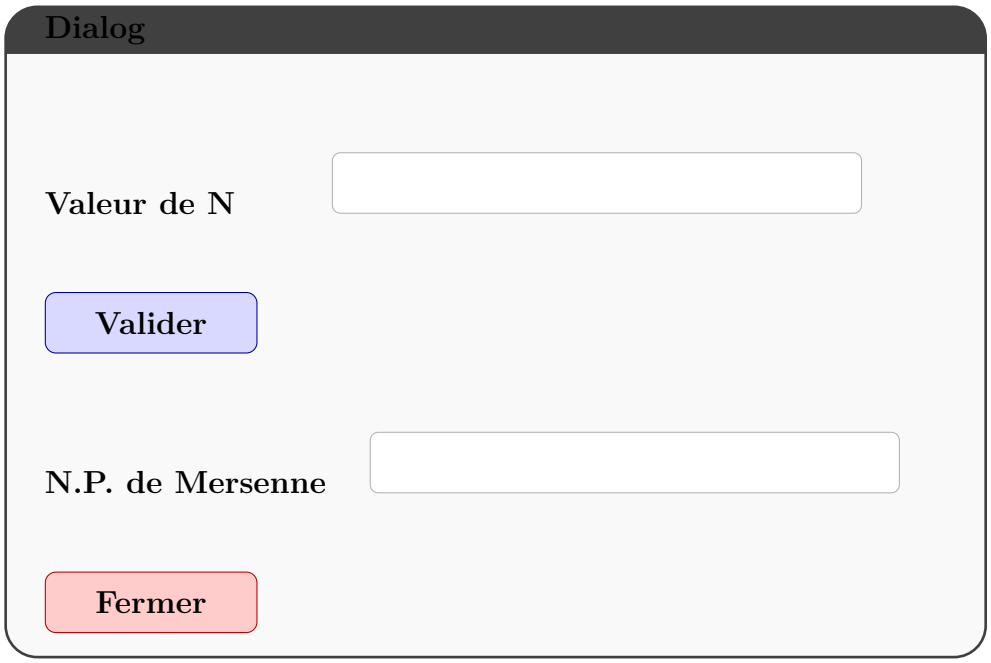
3.3.3 Bouton 3 : Test de primalité



Le fonctionnement du bouton **test de primalité** se déroule selon les étapes suivantes :

1. L'utilisateur saisit un nombre entier N dans le champ intitulé **Valeur N**.
2. Il clique ensuite sur le bouton **Valider**.
3. Le programme utilise alors la fonction `isprime()` de la bibliothèque `sympy` pour tester si N est un nombre premier.
4. Le résultat du test (premier ou non premier) est affiché dans le champ **T.P. simple**.
5. L'utilisateur peut cliquer sur le bouton **Fermer** pour quitter la fenêtre de test.

3.3.4 Bouton 4 : Nombres premiers de Mersenne



Ce bouton ouvre une fenêtre permettant à l'utilisateur de saisir un entier N et de cliquer sur le bouton valider. Le programme calcule automatiquement tous les nombres premiers de Mersenne pour des valeurs n choisies par l'utilisateur (par exemple : de 2 à 20).

3.3.5 Bouton 5 : Fermer

Permet de quitter proprement l'application.

3.4 Exemple de code de l'interface principale avec PyQt5

```
from PyQt5.QtWidgets import QApplication, QWidget,
   QPushButton, QVBoxLayout
import sys

class InterfacePrincipale(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Theorie des Nombres avec
           Python")
        self.setGeometry(100, 100, 400, 300)

        layout = QVBoxLayout()
        boutons = ["Verifier primalite", "Factoriser", "
           Test avance", "Mersenne", "Quitter"]

        for nom in boutons:
            btn = QPushButton(nom)
            btn.clicked.connect(lambda _, texte=nom:
               print(f"Bouton '{texte}' clique"))
            layout.addWidget(btn)

        self.setLayout(layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = InterfacePrincipale()
    window.show()
    sys.exit(app.exec_())
```

3.5 Explication du code

— **Importation des modules :** PyQt5 est utilisé pour gérer les composants graphiques.

— **Création de la fenêtre principale :** la classe `Interface principale` définit une fenêtre contenant cinq boutons.

— **Agencement des boutons :** un layout vertical (`QVBoxLayout`) est utilisé pour disposer les boutons de manière ordonnée.

— **Exécution de l'application :** la fenêtre est affichée via `window.show()` et la boucle événementielle est lancée avec `app.exec_()`.

Conclusion

Ce travail met en évidence l'intérêt d'intégrer la programmation dans l'enseignement des mathématiques, en particulier dans un domaine aussi fondamental que la théorie des nombres. Grâce à Python et PyQt5, il devient possible de rendre vivants des concepts qui paraissent souvent abstraits pour les élèves.

L'approche algorithmique adoptée favorise l'expérimentation, développe la logique, et rend les mathématiques plus concrètes et accessibles. L'interface graphique conçue permet une utilisation intuitive, ce qui en fait un outil idéal pour l'enseignement, l'auto-apprentissage, ou les ateliers scientifiques.

En conclusion, ce projet représente un exemple concret de pédagogie active, où les mathématiques rencontrent la technologie pour renforcer la compréhension, la motivation et l'autonomie des apprenants.

Bibliographie

[1] Python Software Foundation, *Python Language Reference*, Version 3.12, `https://www.python.org`, consulté en 2025.

[2] A. Meurer et al., *SymPy : Symbolic computing in Python*, PeerJ Computer Science, 2017, `https://www.sympy.org`.

[3] Riverbank Computing, *PyQt5 Documentation*, `https://www.riverbankcomputing.com/software/pyqt/intro`, consulté en 2025.

[4] *Arithmétique dans* $\mathbb{N}$. `https://www.alloschool.com`.

[5] The Prime Pages, *Mersenne Primes : History, Theorems and Records*, `https://primes.utm.edu/mersenne/`, consulté en 2025.