



# Introduction

## 0.1. Buts et moyens

Comme son nom l'indique, ce livre a pour but d'apprendre le langage Python en programmant principalement des jeux, qui serviront de prétexte pour découvrir les différentes facettes du langage Python. La difficulté sera progressive : on commencera par des jeux très simples et sans fioritures, pour aller vers des jeux de plus en plus « jolis » et sophistiqués. Chaque jeu permettra d'introduire de nouvelles notions, tout en répétant celles déjà vues auparavant.

## 0.2. Programmer ?

La **programmation** consiste à « expliquer » en détails à un ordinateur ce qu'il doit faire, sachant qu'il ne comprend évidemment pas une langue humaine, mais seulement ce qu'on appelle un **langage de programmation** (par exemple C, Python, Java, etc.). En d'autres termes, il faudra traduire une idée simple (par exemple « trier des nombres dans l'ordre croissant »), en un vrai raisonnement parfaitement structuré et détaillé, que l'on appelle un **algorithme**.

Un langage de programmation a de nombreux points communs avec un langage humain : il a une syntaxe (l'orthographe des mots) et une grammaire (la façon d'agencer les mots). La différence la plus importante est qu'un langage informatique ne tolère **aucune** erreur, alors que l'on peut mal parler une langue tout en se faisant comprendre quand même.



### 0.2.1. Types d'erreurs

La programmation est une tâche complexe, et on y commet de nombreuses erreurs.

#### Erreurs de syntaxe

Un programme ne peut être exécuté que si sa syntaxe est parfaitement correcte. Dans le cas contraire, le processus s'arrête (on parle communément de « plantage ») et vous obtenez un message d'erreur. Le terme syntaxe se réfère aux règles que les auteurs du langage ont établies pour la structure du programme. **Tous** les détails ont de l'importance : le respect des majuscules et des minuscules, l'orthographe, la ponctuation, ...

#### Erreurs sémantiques

Le second type d'erreur est l'erreur sémantique ou erreur de logique. S'il existe une erreur de ce type dans un de vos programmes, il n'y aura aucun message d'erreur, mais le résultat ne sera pas celui que vous attendiez.

#### Erreurs à l'exécution

Le troisième type d'erreur est l'erreur en cours d'exécution (*Run-time error*), qui apparaît seulement lorsque votre programme fonctionne déjà, mais que des circonstances particulières se

Pour des raisons anecdotiques, les erreurs de programmation s'appellent des « bugs ». *bug* est à l'origine un terme anglais servant à désigner de petits insectes gênants, tels les punaises. Il est arrivé à plusieurs reprises que des cadavres de ces insectes provoquent des court-circuits et donc des pannes incompréhensibles.

présentent (par exemple, votre programme essaie de lire un fichier qui n'existe plus).



## Recherche des erreurs et expérimentation

**Débuguer** efficacement un programme demande beaucoup de perspicacité et ce travail ressemble à une enquête policière. Vous examinez les résultats, et vous devez émettre des hypothèses pour reconstituer les processus et les événements qui ont logiquement entraîné ces résultats.

### 0.2.2. Les 4 règles d'or pour bien programmer

Programmer ne suffit pas. Il faut **bien** programmer, de sorte que quelqu'un qui voudra réutiliser votre programme puisse le faire facilement. En un mot, le programme doit être **compréhensible**. La première chose consistera à **décomposer un problème compliqué en plusieurs sous-problèmes simples**, donc à découper le programme en plusieurs sous-programmes.

Voici les règles d'or à respecter :



#### Règle numéro 1.

Ne pas écrire de longs sous-programmes (pas plus de 10-15 lignes de code).



#### Règle numéro 2.

Chaque sous-programme doit avoir un objectif clair.



#### Règle numéro 3.

Écrire des commentaires pour expliquer les parties les plus subtiles du programme.



#### Règle numéro 4 (jamais respectée).

Éviter le copier/coller, source d'innombrables erreurs.

Vous trouverez sur le site web compagnon (voir § 0.4) un extrait des PEP8 PEP257 (Python Extension Proposal) originaux qui reprend les points importants pour l'apprentissage des bonnes méthodes de programmation en Python (voir § 0.3). Les auteurs originaux sont Guido **van Rossum** et Barry **Warsaw**.

### 0.2.3. Citations de sagesse

« L'enseignement de l'informatique ne peut faire de personne un programmeur expert plus que l'étude des pinceaux et du pigment peut faire de quelqu'un un peintre expert. » - Eric S. Raymond

« Programmer, c'est comme se donner des coups de pied dans le visage, tôt ou tard, votre nez va saigner. » - Kyle Woodbury

« Je ne suis pas un excellent programmeur. Je suis juste un bon programmeur avec d'excellentes habitudes. » - Kent Beck

« N'importe quel idiot peut écrire du code qu'un ordinateur peut comprendre. Les bons programmeurs écrivent du code que les humains peuvent comprendre. » - Martin Fowler

« La vérité ne peut être trouvée qu'à un endroit : le code. » - Robert C. Martin



Guido von Rossum

## 0.3. Python

Python est un langage portable, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido **van Rossum** et de nombreux contributeurs bénévoles.

Il est apprécié par les pédagogues qui y trouvent un langage où la syntaxe permet une initiation aisée aux concepts de base de la programmation. Enfin, le langage Python gagne en popularité car il est le langage favori des « data scientists », notamment. Il permet également de faire du Web aisément avec *Django*.

### 0.3.1. Principales caractéristiques du langage

Détaillons quelques caractéristiques de Python :

- Python est **portable** : il fonctionne non seulement sur Linux, mais aussi sur MacOS, et les différentes variantes de Windows.
- Python est **gratuit**, mais on peut l'utiliser sans restriction dans des projets commerciaux.
- Python convient aussi bien à des **scripts** d'une dizaine de lignes qu'à des **projets complexes** de plusieurs dizaines de milliers de lignes.
- La **syntaxe** de Python est très simple et, combinée à des **types de données évolués** (listes, dictionnaires,...), conduit à des programmes à la fois très compacts et très lisibles. À fonctionnalités égales, un programme Python (abondamment commenté et présenté selon les canons standards) est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.
- Python est **orienté objet**. Il supporte l'**héritage multiple** et la **surcharge des opérateurs**.
- Python est un langage qui **continue à évoluer**, soutenu par une communauté d'utilisateurs enthousiastes et responsables, dont la plupart sont des supporters du logiciel libre. Nous utiliserons dans ce cours la **version 3**. Il est à noter qu'un programme écrit en Python 2 n'est pas compatible avec la version 3, et nécessitera quelques modifications.

### 0.3.2. Installation

Nous allons utiliser *EduPython*, qui est un environnement de développement Python gratuit permettant une prise en main rapide, et surtout qui intègre plusieurs packages qui pourront nous être utiles : Numpy, Scipy, Mathplotlib, PIL, Pygame, etc.

Allez sur le site <https://edupython.tuxfamily.org/> et installez la dernière version.

## 0.4. Site web compagnon

Vous trouverez sur le site associé à ce cours les programmes des jeux, afin d'éviter de perdre du temps à les recopier et afin de pouvoir les tester facilement. Vous trouverez aussi les corrigés des exercices et les ressources à utiliser.

L'adresse est [www.apprendre-en-ligne.net/pj/](http://www.apprendre-en-ligne.net/pj/)

## 0.5. Contenu du cours

Dans ce cours, les jeux et autres amusements auront une place prépondérante, même si, de temps à autre, surtout dans les premiers chapitres, on trouvera des exercices concernant d'autres sujets.

### Première partie : bases de Python

Au chapitre 1, nous aborderons un élément essentiel des jeux : le **hasard**. Nous aborderons du même coup en douceur quelques concepts importants de la programmation : l'affichage, les opérateurs, les boucles et les conditions.

Au chapitre 2, le jeu **Devine mon nombre** permettra de voir deux concepts fondamentaux de la programmation : les conditions et les boucles. Il sera aussi question de variables. On calculera plus loin les probabilités de perdre des armées lors de combats au **Risk**.

Au chapitre 3, on jouera à **Pierre, papier, ciseaux** contre l'ordinateur, ce qui nous permettra d'introduire les procédures et les fonctions.

On retrouvera ce jeu au chapitre 4, mais cette fois dans une version graphique. On calculera aussi des probabilités avec **Let's make a deal !**

Au chapitre 5, nous découvrirons le **jeu de Juniper Green**, où nous devons utiliser des listes. Nous analyserons aussi le célèbre jeu pour enfants **Le verger**.

Au chapitre 6, nous aborderons un des piliers de la théorie des jeux : le **dilemme du prisonnier**, qui nous permettra de découvrir les dictionnaires et les fonctions lambda.

Au chapitre 7, nous utiliserons plusieurs jeux de lettres : **le pendu**, **le mot le plus long**, **le Scrabble** et **Motus**, ce qui nous amènera tout naturellement à étudier les fichiers et les chaînes de caractères.

### Deuxième partie : graphisme et animation

Nous programmerons une version graphique du pendu au chapitre 8.

Dans le chapitre 9, il faudra écrire un programme qui nous permettra de jouer au **Memory** contre l'ordinateur.

Dans le chapitre 10, nous utiliserons le **Blackjack** pour introduire l'importante notion de programmation orientée objet.

Le chapitre 11, exceptionnellement, ne concernera pas directement les jeux, mais sera quand même amusant : il sera en effet question de dessins et nous verrons comment utiliser des visages stylisés pour représenter des données statistiques.

Au chapitre 12, nous verrons comment bouger des images lors d'une **course d'escargots**, et d'un **Space Invaders** simplifié. Nous retrouverons aussi avec nostalgie le premier jeu vidéo à avoir connu un succès populaire : **Pong**. Finalement, nous verrons comment simuler une **planche de Galton**.

Qui dit jeux dit souvent damiers. Au chapitre 13, nous approfondirons le chapitre précédent en dessinant différents damiers (qui nous permettront de programmer différentes questions concernant **les échecs**) et nous nous prendrons un moment pour le peintre Vasarely.

Le chapitre 14 traitera d'un classique de la programmation pour débutants : les automates cellulaires, et en particulier le célèbre **jeu de la vie** de **Conway**. Nous programmerons aussi le casse-tête **Gasp**.

Au chapitre 15, nous aborderons la récursivité avec le célèbre jeu du **démineur**. Nous verrons aussi comment **sortir d'un labyrinthe**.

Le chapitre 16 sera la cerise sur le gâteau. En effet, nous allons programmer un **jeu d'échecs**. En fait, pas tout à fait. Nous allons faire ce qui se fait très souvent en informatique : partir d'un programme (relativement simple) disponible sur le web, puis l'améliorer. Cela demandera

évidemment de le comprendre avant de le retoucher... Nous créerons dans un premier temps une interface graphique, puis, dans un deuxième temps, nous essaierons de le rendre plus fort.

## 0.6. Remarques sur les exercices

Ce livre est destiné à l'enseignement dans des classes. L'une des difficultés de l'enseignement de l'informatique est que chaque élève a son propre rythme, et que les niveaux des élèves lors des premières leçons peuvent être très différents. Cela signifie qu'après quelques leçons déjà, certains élèves peuvent avoir plusieurs chapitres d'avance sur les plus lents. Pour y remédier, il faut avoir quelques exercices supplémentaires pour occuper les plus rapides.

En face de chaque exercice, vous trouverez une de ces trois icônes :



indique un exercice relativement facile que tous les élèves devront faire, si possible sans ChatGPT (voir § 0.7).



indique un exercice que tous les élèves devront faire, mais qui est un peu plus difficile. S'ils y passent trop de temps, ils auront avantage à utiliser ChatGPT ou à regarder le corrigé.



indique un exercice difficile que les élèves les plus rapides pourront faire s'ils le souhaitent, en attendant les élèves les plus lents.

De plus, 256 exercices plus courts sont disponibles dans le **Défi Turing**, qui propose des problèmes de maths où l'informatique sera souvent nécessaire.

L'adresse est [www.apprendre-en-ligne.net/turing/](http://www.apprendre-en-ligne.net/turing/)

Presque toutes les bases théoriques pour faire ces exercices seront acquises à la fin du chapitre 7.



## 0.7. ChatGPT et consorts

Depuis l'avènement de ChatGPT, l'enseignement de la programmation doit évoluer. En effet, la plupart des exercices de ce cours peuvent être faits en quelques secondes par ChatGPT. Le problème est que le vrai but n'est pas de faire l'exercice le plus rapidement possible, mais de concevoir un plan d'action (un algorithme) et le traduire dans le langage Python.

Alors comment utiliser ChatGPT (ou un concurrent comme Claude par exemple) pour apprendre à programmer ? J'ai posé la question à... ChatGPT et voici sa réponse :

« Utiliser ChatGPT pour enseigner Python peut être une expérience enrichissante si les élèves apprennent à l'utiliser comme un outil de réflexion, d'exploration et de validation plutôt que comme une simple machine à résoudre des problèmes. Voici une approche structurée pour maximiser son potentiel pédagogique :

### 1. Définir les rôles de ChatGPT

- **Un guide, pas une béquille** : Encouragez les élèves à poser des questions ciblées pour débloquer leur réflexion, plutôt que de demander directement des solutions complètes.
- **Un mentor, pas un correcteur** : ChatGPT peut expliquer des concepts, donner des pistes ou des indices, mais les élèves doivent d'abord essayer de résoudre les problèmes eux-

mêmes.

## 2. Intégrer ChatGPT dans les exercices

### 1. Phase de découverte : Comprendre les concepts

- **Exemple :** "Qu'est-ce qu'une boucle `for` en Python et quand l'utiliser ?"
- ChatGPT peut fournir des explications adaptées et des exemples simples.
- Les élèves comparent ces explications avec leur cours ou leurs notes pour s'assurer de la compréhension.

### 2. Phase d'exploration : Tester des idées

- **Exemple :** "Peux-tu m'expliquer pourquoi mon code plante avec cette erreur ?"
- Les élèves présentent leur code et analysent les suggestions fournies par ChatGPT.
- Ils peuvent demander des variations : "Peux-tu écrire un exemple similaire mais en utilisant une liste ?"

### 3. Phase de production : Écrire leur propre code

- Les élèves écrivent leur code en autonomie.
- Ils utilisent ChatGPT pour demander des clarifications spécifiques, par exemple : "Comment lire un fichier ligne par ligne en Python ?"

### 4. Phase de validation : Vérifier et améliorer

- Les élèves demandent à ChatGPT d'examiner leur code :
  - "Peux-tu vérifier si mon code est optimisé ?"
  - "Y a-t-il une manière plus élégante de résoudre ce problème ?"
- Ils comparent les solutions proposées avec la leur.

## 3. Développer la pensée critique

- **Comparer les solutions :** Proposez aux élèves de demander une solution à ChatGPT, puis de l'évaluer :
  - Est-elle compréhensible ?
  - Est-elle plus ou moins efficace que la leur ?
  - Quels en sont les avantages et inconvénients ?
- **Exemple de réflexion :**
  - ChatGPT propose une compréhension différente d'un problème : "Pourquoi ChatGPT a-t-il utilisé une boucle `while` au lieu d'une boucle `for` dans cet exemple ?"

## 4. Encourager l'apprentissage par l'erreur

Demandez aux élèves d'utiliser ChatGPT pour simuler des erreurs courantes et les corriger.

- **Exemple :** "Quel type d'erreur pourrait survenir si j'essaie d'accéder à un index qui n'existe pas dans une liste ?"
- Les élèves écrivent du code qui provoque cette erreur, la testent, puis utilisent ChatGPT pour en comprendre les causes et solutions.

## 5. Stimuler la créativité

- **Jeux et défis :**
  - "Demandez à ChatGPT de générer une idée de projet simple, puis réalisez-le en groupe."
  - Exemple : "Créer un programme qui génère un mot de passe aléatoire sécurisé."
- **Exercices inversés :** Donnez un code produit par ChatGPT et demandez aux élèves d'en décrire le fonctionnement ou de l'optimiser.

## 6. Enseigner les limites de ChatGPT

- **Fiabilité des réponses :** Montrez que ChatGPT peut se tromper ou produire des solutions non optimales. Encouragez les élèves à :
  - Vérifier les réponses avec des sources fiables (documentation officielle, cours).
  - Tester le code pour confirmer son fonctionnement.
- **Formuler les bonnes questions :** Apprenez aux élèves à poser des questions claires et précises pour obtenir des réponses utiles.

## 7. Évaluer les compétences des élèves

- Créez des exercices où l'usage de ChatGPT est limité :
  - Résoudre un problème sans assistance directe.
  - Expliquer un concept ou une solution sans copier-coller.
- Demandez aux élèves de documenter comment ils ont utilisé ChatGPT pour résoudre un problème, en expliquant :
  - Ce qu'ils ont demandé.
  - Ce qu'ils ont appris grâce à l'interaction.

En suivant cette méthodologie, vos élèves apprendront non seulement à programmer en Python, mais aussi à exploiter intelligemment les outils numériques comme ChatGPT pour devenir des développeurs autonomes et critiques. »



## 0.8. Ce que vous avez appris dans l'introduction

- Ce qu'est la programmation et quels sont les types d'erreurs que vous rencontrerez.
- Les 4 règles d'or pour bien programmer.
- Les principales caractéristiques de Python 3 et comment l'installer.
- Comment on utilisera ChatGPT pour apprendre à programmer.