Livret de révision de terminale

Ce livret reprend, chapitre par chapitre, ce qu'il faut savoir et savoir faire en fin de terminale. Les algorithmes eux-mêmes, avec leur code commenté, sont dans le livret « Algorithmes du programme ».

① Dictionnaires

Ce qu'il faut savoir

— Un dictionnaire associe des **clés** à des **valeurs** : chaque élément est une paire `clé: valeur`. Les clés sont uniques et doivent être d'un type non modifiable (nombre, chaîne de caractères, tuple).

— On accède à une valeur par sa clé, avec la notation `d[cle]`, et non par une position.

— En Python, un dictionnaire est implémenté par une **table de hachage** : l'accès à une valeur par sa clé, l'ajout et le test d'appartenance `cle in d` se font en temps constant en moyenne, quel que soit le nombre d'éléments.

Ce qu'il faut savoir faire

— Créer un dictionnaire vide (`d = {}`) ou déjà rempli (`d = {"nom": "Dupont", "age": 17}`).

— Ajouter ou modifier une valeur (`d["ville"] = "Lyon"`), supprimer une paire (`del d["age"]`), lire une valeur avec une valeur par défaut (`d.get("ville", "inconnue")`).

— Parcourir les clés, les valeurs ou les paires avec `d.keys()`, `d.values()` et `d.items()`.

```
notes = {"Alice": 15, "Ben": 12}
for nom, note in notes.items():
    print(f"{nom} : {note}")
```

② Récursivité

Ce qu'il faut savoir

— Une **fonction récursive** est une fonction qui s'appelle elle-même.

— Elle comporte au moins un **cas de base**, traité sans appel récursif, et des appels récursifs sur des données plus petites, qui finissent par atteindre un cas de base. Sans cas de base atteint, les appels ne s'arrêtent jamais.

— Chaque appel est placé sur la **pile d'appels** ; Python limite sa taille (environ 1000 appels) et signale `RecursionError` au-delà.

— Une définition récursive suit souvent une relation de récurrence mathématique, comme $n! = n \times (n-1)!$ avec $0! = 1$.

Ce qu'il faut savoir faire

— Écrire une fonction récursive simple en identifiant le cas de base et l'appel récursif.

— Dérouler à la main les appels d'une fonction récursive.

```python
def factorielle(n):
    if n == 0:
        return 1
    return n * factorielle(n - 1)
```

③ Programmation orientée objet

Ce qu'il faut savoir

— Une **classe** décrit un modèle d'objets ; un **objet**, ou **instance**, est créé à partir de ce modèle.

— Les **attributs** représentent l'état d'un objet, les **méthodes** son comportement.

— La méthode `__init__`, appelée le **constructeur**, initialise les attributs à la création de l'objet. Le premier paramètre de chaque méthode, `self`, désigne l'objet sur lequel la méthode est appelée.

— L'héritage n'est pas au programme.

Ce qu'il faut savoir faire

— Écrire une classe avec son constructeur, ses attributs et ses méthodes.

— Créer des instances, lire et modifier leurs attributs, appeler leurs méthodes.

— Lire et comprendre un programme qui utilise des classes.

```python
class Compte:
    def __init__(self, titulaire, solde=0):
        self.titulaire = titulaire
        self.solde = solde

    def deposer(self, montant):
        self.solde = self.solde + montant

c = Compte("Alice")
c.deposer(50)
```

④ Modularité

Ce qu'il faut savoir

— Un **module** est un fichier Python qui regroupe des définitions réutilisables ; on l'importe avec `import module` ou `from module import fonction`.

— L'**interface** (ou API) d'un module est l'ensemble des fonctions qu'il met à disposition, avec leur documentation ; l'utilisateur n'a pas besoin de connaître leur **implémentation**.

— Découper un programme en modules le rend plus lisible, plus facile à tester et à réutiliser, et permet de travailler à plusieurs.

Ce qu'il faut savoir faire

— Utiliser un module ou une bibliothèque à partir de sa documentation (`help(fonction)`).

— Documenter une fonction par une **docstring**.

— Créer un module et l'importer dans un autre programme.

5 Programmation fonctionnelle

Ce qu'il faut savoir

— La programmation fonctionnelle est un **paradigme** de programmation, comme la programmation impérative ou la programmation orientée objet.

— Une **fonction pure** renvoie toujours le même résultat pour les mêmes arguments et ne produit aucun **effet de bord** : elle ne modifie ni ses arguments ni les variables extérieures.

— En Python, les fonctions sont des valeurs comme les autres : on peut les affecter à une variable, les passer en argument à une autre fonction ou les renvoyer. Une fonction qui prend ou renvoie une fonction est dite **d'ordre supérieur**.

— Le mot-clé `lambda` définit une fonction anonyme courte, par exemple `lambda x: 2 * x`.

Ce qu'il faut savoir faire

— Écrire des fonctions sans effet de bord.

— Utiliser `map`, `filter` et le paramètre `key` de `sorted`.

```
L = [3, -1, 8, -5]
positifs = list(filter(lambda x: x > 0, L))
doubles = list(map(lambda x: 2 * x, L))
```

6 Mise au point des programmes

Ce qu'il faut savoir

— Une erreur dans un programme peut avoir des conséquences graves : en 1996, la fusée Ariane 5 a été détruite en vol parce qu'un nombre flottant sur 64 bits avait été converti en un entier sur 16 bits trop petit pour le contenir.

— Une **précondition** est une condition que les arguments doivent vérifier avant l'appel d'une fonction ; une **postcondition** est une propriété que le résultat doit vérifier. On peut les contrôler avec `assert`.

— Un **test** vérifie le résultat d'une fonction sur un exemple dont on connaît la réponse. Les tests montrent la présence d'erreurs, jamais leur absence.

— Erreurs fréquentes : mauvais type de donnée, indice hors de la liste, condition oubliée dans un `if`, inégalité stricte au lieu de large, comparaison d'égalité entre flottants, effet de bord non voulu.

— Un **débogueur** permet d'exécuter un programme pas à pas, de placer des points d'arrêt et d'observer les variables.

Ce qu'il faut savoir faire

— Écrire une docstring, des assertions et des tests (`doctest` ou `assert`).

— Lire un message d'erreur Python (`TypeError`, `IndexError`, `NameError`, etc.) et retrouver la ligne en cause.

7 Listes chaînées, piles et files

Ce qu'il faut savoir

— Un type abstrait se définit par son **interface**, c'est-à-dire les opérations qu'il propose, indépendamment de son **implémentation**.

— Une **liste chaînée** est formée de maillons, ou nœuds, qui contiennent chacun une valeur et une référence vers le maillon suivant. Ajouter ou retirer en tête se fait en temps constant ; accéder au i-ième élément demande de parcourir les i premiers maillons.

— Une **pile** fonctionne selon le principe « dernier entré, premier sorti » (en anglais LIFO) : on **empile** et on **dépile** au sommet.

— Une **file** fonctionne selon le principe « premier entré, premier sorti » (en anglais FIFO) : on **enfile** à une extrémité et on **défile** à l'autre.

— Opérations de base d'une pile ou d'une file : créer une structure vide, tester si elle est vide, ajouter un élément, retirer un élément.

Ce qu'il faut savoir faire

— Implémenter une liste chaînée, une pile et une file avec une classe.

— Choisir la structure adaptée à un problème (historique de navigation : pile ; file d'attente d'impression : file).

8 Arbres

Ce qu'il faut savoir

— Un **arbre** est une structure hiérarchique formée de **nœuds** reliés par des **arêtes**. La **racine** est le seul nœud sans père ; une **feuille** est un nœud sans fils.

— La **taille** d'un arbre est son nombre de nœuds. La **profondeur** d'un nœud est le nombre d'arêtes entre la racine et ce nœud. La **hauteur** d'un arbre est la plus grande profondeur de ses feuilles : un arbre réduit à sa racine a une hauteur de 0, l'arbre vide une hauteur de -1.

— Dans un **arbre binaire**, chaque nœud a au plus deux fils, appelés fils gauche et fils droit. Un arbre binaire de taille n et de hauteur h vérifie $\lfloor \log_2 n \rfloor \leqslant h \leqslant n - 1$.

— Un **arbre binaire de recherche** est un arbre binaire dans lequel, pour chaque nœud, les clés du sous-arbre gauche sont inférieures à la clé du nœud et celles du sous-arbre droit lui sont supérieures.

— Rechercher ou insérer une clé dans un arbre binaire de recherche demande un nombre d'étapes proportionnel à sa hauteur : de l'ordre de $\log_2 n$ si l'arbre est équilibré, de l'ordre de n s'il est filiforme.

Ce qu'il faut savoir faire

— Calculer récursivement la taille et la hauteur d'un arbre binaire.

— Parcourir un arbre binaire en profondeur, dans les ordres **préfixe**, **infixe** et **suffixe**, et en largeur.

— Rechercher et insérer une clé dans un arbre binaire de recherche.

9 Graphes

Ce qu'il faut savoir

— Un **graphe** est formé de **sommets** reliés par des **arêtes** ; dans un graphe **orienté**, les liens ont un sens et s'appellent des **arcs**. Un graphe est **pondéré** quand chaque arête porte une valeur, son poids.

— Un **chemin** (ou chaîne) est une suite de sommets reliés deux à deux par une arête ; un **cycle** est un chemin qui revient à son point de départ.

— Un graphe se représente par une **matrice d'adjacence**, tableau qui contient 1 à la ligne i et à la colonne j quand les sommets i et j sont reliés, ou par des **listes d'adjacence**, dictionnaire qui associe à chaque sommet la liste de ses voisins. La matrice convient aux graphes où presque tous les sommets sont reliés, les listes aux graphes qui ont peu d'arêtes.

— Le **parcours en largeur** visite les sommets par distance croissante au départ, à l'aide d'une **file** ; le **parcours en profondeur** va le plus loin possible avant de revenir en arrière, à l'aide d'une **pile** ou de la récursivité.

Ce qu'il faut savoir faire

— Passer d'un dessin de graphe à sa matrice ou à ses listes d'adjacence, et inversement.

— Parcourir un graphe en largeur et en profondeur, repérer la présence d'un cycle, chercher un chemin entre deux sommets.

10 Diviser pour régner

Ce qu'il faut savoir

— La méthode **diviser pour régner** résout un problème en trois étapes : **diviser** le problème en sous-problèmes plus petits, **régner** en résolvant ces sous-problèmes, souvent par des appels récursifs, puis **combiner** leurs solutions.

— Le **tri fusion** coupe la liste en deux moitiés, trie chaque moitié récursivement, puis fusionne les deux moitiés triées. Il trie n éléments en un temps de l'ordre de $n \log_2 n$, bien moins que les tris par sélection ou par insertion, de l'ordre de n^2.

— La recherche dichotomique dans une liste triée suit la même idée : elle élimine la moitié des éléments à chaque étape.

Ce qu'il faut savoir faire

— Écrire la fusion de deux listes triées, puis le tri fusion.

— Reconnaître les trois étapes dans un algorithme donné.

11 Programmation dynamique

Ce qu'il faut savoir

— La **programmation dynamique** résout un problème à partir des solutions de sous-problèmes qui se répètent, en **mémorisant** chaque résultat pour ne le calculer qu'une fois.

— Deux façons de faire : de haut en bas, avec une fonction récursive et un dictionnaire de résultats déjà calculés (**mémoïsation**) ; de bas en haut, en remplissant un tableau des plus petits sous-problèmes aux plus grands.

— Exemple : le calcul récursif naïf des termes de la suite de Fibonacci refait les mêmes calculs un nombre exponentiel de fois ; avec mémorisation, chaque terme n'est calculé qu'une fois.

— Contrairement à un algorithme glouton, la programmation dynamique donne toujours une solution optimale au rendu de monnaie ou au problème du sac à dos.

Ce qu'il faut savoir faire

— Écrire une version mémoïsée d'une fonction récursive.

— Résoudre un problème simple (rendu de monnaie, sac à dos, plus long chemin dans une grille) en remplissant un tableau.

12 Recherche textuelle

Ce qu'il faut savoir

— Il s'agit de trouver toutes les positions d'un **motif** dans un **texte**.

— L'**algorithme naïf** compare le motif à chaque position du texte et décale toujours d'un seul caractère.

— L'**algorithme de Boyer-Moore**, ici dans sa version de Horspool, compare le motif au texte **de droite à gauche**. Il calcule à l'avance, pour chaque caractère du motif, le décalage à appliquer en cas d'échec ; un caractère absent du motif permet de décaler du motif entier.

— Plus le motif est long, plus les décalages sont grands, et plus Boyer-Moore est rapide par rapport à l'algorithme naïf.

Ce qu'il faut savoir faire

— Construire la table des décalages d'un motif.

— Dérouler l'algorithme de Boyer-Moore-Horspool à la main sur un exemple.

13 Bases de données relationnelles

Ce qu'il faut savoir

— Dans le **modèle relationnel**, les données sont rangées dans des **relations** (ou tables). Chaque ligne d'une relation est un **enregistrement**, chaque colonne un **attribut**, avec son **domaine**, c'est-à-dire l'ensemble des valeurs qu'il peut prendre.

— La **clé primaire** est un attribut, ou un groupe d'attributs, dont la valeur identifie chaque enregistrement de façon unique.

— Une **clé étrangère** est un attribut qui fait référence à la clé primaire d'une autre relation ; elle relie les deux relations.

— Les **contraintes d'intégrité** (de domaine, de clé primaire, de référence) empêchent d'enregistrer des données incohérentes, par exemple une clé étrangère qui ne correspond à aucun enregistrement.

— Le **schéma relationnel** liste les relations, leurs attributs avec leur domaine, la clé primaire (soulignée) et les clés étrangères (précédées de #).

— Un **système de gestion de bases de données** (SGBD), comme SQLite, MySQL ou PostgreSQL, assure la persistance des données, gère les accès simultanés de plusieurs utilisateurs, contrôle les droits d'accès et traite les requêtes efficacement.

Ce qu'il faut savoir faire

— Lire et écrire un schéma relationnel.

— Repérer les clés primaires et étrangères, et les contraintes d'intégrité qu'une modification violerait.

14 Langage SQL

Ce qu'il faut savoir

Les exemples portent sur deux relations :

— `Auteurs(id, nom, prenom, naissance)` ;

— `Livres(id, titre, #id_auteur, annee, note)`.

```sql
-- sélection, filtre, tri
SELECT titre, annee FROM Livres WHERE note >= 8 ORDER BY annee DESC;
-- valeurs distinctes et fonctions d'agrégation
SELECT DISTINCT annee FROM Livres;
SELECT COUNT(*), AVG(note), MAX(note) FROM Livres;
-- jointure
SELECT Livres.titre, Auteurs.nom
FROM Livres JOIN Auteurs ON Livres.id_auteur = Auteurs.id;
-- modification des données
INSERT INTO Livres VALUES (12, '1984', 3, 1949, 10);
UPDATE Livres SET note = 7 WHERE titre = 'Hypérion';
DELETE FROM Livres WHERE annee < 1900;
```

Ce qu'il faut savoir faire

— Écrire une requête de sélection avec `WHERE`, `ORDER BY`, `DISTINCT` et les fonctions `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.

— Écrire une jointure entre deux tables avec `JOIN ... ON`.

— Insérer, modifier et supprimer des données avec `INSERT`, `UPDATE` et `DELETE`.

15 Processus

Ce qu'il faut savoir

— Un **processus** est un programme en cours d'exécution. Le système d'exploitation lui attribue un identifiant, le **PID** ; le **PPID** est le PID du processus parent qui l'a créé.

— Un processus est **prêt** quand il attend son tour, **élu** quand il s'exécute sur le processeur, **bloqué** quand il attend une ressource (une saisie, un fichier, une donnée réseau).

— L'**ordonnanceur** du système d'exploitation choisit à chaque instant le processus prêt qui devient élu ; en alternant très vite, il donne l'impression que les processus s'exécutent en même temps.

— Il y a **interblocage** quand plusieurs processus s'attendent mutuellement : chacun détient une ressource dont un autre a besoin, et aucun ne peut avancer.

Ce qu'il faut savoir faire

— Lister les processus avec `ps -ef` ou `top`, en arrêter un avec `kill PID`.

— Dérouler un ordonnancement simple et repérer une situation d'interblocage.

16 Systèmes sur puce

Ce qu'il faut savoir

— Un **système sur puce** (en anglais SoC, pour *System on Chip*) réunit sur une seule puce de quelques centimètres carrés les composants d'un ordinateur : processeur, mémoire, processeur graphique, circuits de communication (Wi-Fi, Bluetooth, 4G).

— Les composants étant très proches, les données circulent plus vite qu'entre les cartes d'un ordinateur classique, et la puce consomme beaucoup moins d'énergie ; elle chauffe peu et se passe souvent de ventilateur.

— On en trouve dans les téléphones, les tablettes, les consoles portables, les nano-ordinateurs comme le Raspberry Pi, et depuis 2020 dans des ordinateurs portables.

Ce qu'il faut savoir faire

— Identifier les principaux composants sur un schéma de système sur puce.

— Expliquer les avantages de leur intégration en termes de vitesse et de consommation.

17 Protocoles de routage

Ce qu'il faut savoir

— Un **routeur** relie plusieurs réseaux et transmet chaque paquet vers sa destination à l'aide de sa **table de routage**, qui indique pour chaque réseau de destination la passerelle ou l'interface à utiliser et une **métrique**, mesure du coût de la route.

— Un réseau de routeurs se modélise par un graphe : les routeurs sont les sommets, les liaisons les arêtes.

— Le protocole **RIP** est un protocole à vecteur de distances : chaque routeur échange sa table avec ses voisins, et la métrique est le **nombre de sauts**, c'est-à-dire le nombre de routeurs traversés. Il s'appuie sur l'algorithme de Bellman-Ford.

— Le protocole **OSPF** est un protocole à état de liens : chaque routeur connaît la carte complète du réseau et calcule les meilleures routes avec l'algorithme de **Dijkstra**. La métrique est la somme des coûts des liaisons, avec coût $= \dfrac{10^8}{\text{débit en bit/s}}$.

Ce qu'il faut savoir faire

— Compléter la table de routage d'un routeur selon RIP ou selon OSPF.

— Déterminer la route suivie par un paquet avec chacun des deux protocoles.

18 Sécurisation des communications

Ce qu'il faut savoir

— Une communication sécurisée garantit la **confidentialité** (seul le destinataire peut lire le message), l'**intégrité** (le message n'a pas été modifié) et l'**authentification** (on communique bien avec la personne voulue).

— Dans le **chiffrement symétrique**, la même clé secrète sert à chiffrer et à déchiffrer ; les deux interlocuteurs doivent se l'échanger sans qu'un tiers l'intercepte.

— Dans le **chiffrement asymétrique**, chacun possède une **clé publique**, diffusée à tous, et une **clé privée**, gardée secrète. Pour écrire à B, A chiffre avec la clé publique de B ; seul B peut déchiffrer avec sa clé privée.

— Le protocole **HTTPS** utilise le chiffrement asymétrique pour échanger une clé secrète, puis le chiffrement symétrique, plus rapide, pour chiffrer les données ; le **certificat** du serveur, signé par une autorité de certification, authentifie le site.

Ce qu'il faut savoir faire

— Chiffrer et déchiffrer un message avec une clé symétrique simple (code de César, chiffrement par XOR).

— Décrire les étapes d'un échange HTTPS.

19 Calculabilité et décidabilité

Ce qu'il faut savoir

— Un programme est aussi une donnée : un programme peut en lire, en analyser ou en exécuter un autre, comme le fait l'interpréteur Python.

— Un problème est **calculable** quand un algorithme le résout pour toutes ses entrées. Un problème dont la réponse est oui ou non est **décidable** quand un algorithme donne la bonne réponse pour toutes ses entrées.

— La calculabilité ne dépend pas du langage de programmation : tous les langages usuels permettent de calculer les mêmes choses.

— En 1936, Alan Turing a montré que le **problème de l'arrêt** est indécidable : aucun programme ne peut dire, pour tout programme et toute entrée, si ce programme s'arrête.

Ce qu'il faut savoir faire

— Expliquer, sans formalisme, le raisonnement par l'absurde qui montre que le problème de l'arrêt est indécidable.