Livret de révision de première

Ce livret reprend, chapitre par chapitre, ce qu'il faut savoir et savoir faire du programme de première. Il sert à préparer l'entrée en terminale, où toutes ces notions sont supposées acquises.

① Bases de Python

Ce qu'il faut savoir

— Types de base : entiers (`int`), flottants (`float`), booléens (`bool`), chaînes de caractères (`str`). La fonction `type` donne le type d'une valeur.

— Une **affectation** (`x = 3`) associe une valeur à un nom de variable.

— Une instruction conditionnelle choisit un bloc à exécuter avec `if`, `elif` et `else` ; l'indentation délimite les blocs.

— La boucle `for` répète un bloc pour chaque valeur d'une séquence, par exemple `range(n)` qui donne 0, 1, ..., $n-1$; la boucle `while` répète un bloc tant qu'une condition est vraie.

— Une **fonction** se définit avec `def` ; elle reçoit des paramètres et renvoie un résultat avec `return`. Une variable créée dans une fonction est **locale** : elle n'existe pas en dehors.

Ce qu'il faut savoir faire

— Écrire un programme avec des conditions, des boucles et des fonctions.

— Afficher un résultat avec une f-string : `print(f"La moyenne vaut {m}")`.

② Listes, tuples et slicing

Ce qu'il faut savoir

— Une **liste** (`[3, 8, 1]`) et un **tuple** (`(3, 8, 1)`) sont des séquences dont les éléments sont repérés par leur **indice**, de 0 à `len(t) - 1`. L'indice -1 désigne le dernier élément.

— Une liste est modifiable (`L[0] = 5`, `L.append(7)`) ; un tuple ne l'est pas.

— Une liste de listes représente un tableau à deux dimensions : `M[i][j]` est l'élément de la ligne i et de la colonne j.

— Le **slicing** extrait une partie d'une séquence : `L[a:b]` contient les éléments d'indice a à $b-1$.

— Une **liste en compréhension** construit une liste en une ligne : `[x * x for x in L if x > 0]`.

— Après `M = L`, les deux noms désignent la même liste : modifier `M` modifie `L`. Pour obtenir une copie, on écrit `M = L[:]` ou `M = list(L)`.

Ce qu'il faut savoir faire

— Parcourir une liste par ses indices (`for i in range(len(L))`) ou par ses éléments (`for x in L`).

— Construire une liste dans une boucle avec `append`, ou en compréhension.

— Renvoyer plusieurs valeurs d'une fonction sous forme de tuple.

③ Dictionnaires

Ce qu'il faut savoir

— Un dictionnaire associe des **clés** à des **valeurs** : `d = {"nom": "Dupont", "age": 17}`. Les clés sont uniques et non modifiables (nombres, chaînes, tuples).

— On accède à une valeur par sa clé (`d["nom"]`), pas par une position.

Ce qu'il faut savoir faire

— Ajouter ou modifier une valeur (`d["ville"] = "Lyon"`), tester la présence d'une clé (`"ville" in d`), supprimer une paire (`del d["age"]`).

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

— Compter des occurrences avec un dictionnaire.

```
compte = {}
for lettre in "abracadabra":
    if lettre in compte:
        compte[lettre] = compte[lettre] + 1
    else:
        compte[lettre] = 1
```

④ Fichiers et données en tables

Ce qu'il faut savoir

— Un fichier **CSV** est un fichier texte qui représente une table : chaque ligne est un enregistrement, et un séparateur (virgule ou point-virgule) sépare les valeurs des différentes colonnes. La première ligne donne souvent le nom des colonnes.

— On ouvre un fichier avec `open`, de préférence dans un bloc `with`, qui le referme automatiquement.

— Le module `csv` lit une table sous forme de liste de dictionnaires (`csv.DictReader`), un dictionnaire par ligne.

Ce qu'il faut savoir faire

— Lire une table CSV, sélectionner les lignes qui vérifient une condition, trier une table selon une colonne (`sorted(table, key=...)`).

— Fusionner deux tables qui ont une colonne commune.

5 Représentation des entiers

Ce qu'il faut savoir

— Un entier naturel s'écrit en base 2 (binaire), en base 10 ou en base 16 (hexadécimal, chiffres 0 à 9 puis A à F). Avec n bits, on code les entiers de 0 à $2^n - 1$.

— Les entiers relatifs sont codés en **complément à deux** : sur n bits, on code les entiers de -2^{n-1} à $2^{n-1} - 1$, et le bit de poids fort vaut 1 pour les nombres négatifs.

Ce qu'il faut savoir faire

— Convertir un entier d'une base à une autre (2, 10, 16).

— Coder un entier relatif en complément à deux sur n bits : écrire $|x|$ en binaire, inverser tous les bits, ajouter 1. Décoder un code en complément à deux.

6 Représentation des réels

Ce qu'il faut savoir

— Les nombres réels sont représentés en machine par des **flottants**, selon la norme **IEEE 754** : un bit de signe, un exposant et une mantisse (en double précision, sur 64 bits : 1 bit de signe, 11 bits d'exposant, 52 bits de mantisse).

— Beaucoup de nombres décimaux, comme 0,1, n'ont pas d'écriture binaire finie : le flottant n'en est qu'une valeur approchée. Ainsi `0.1 + 0.2 == 0.3` vaut `False`.

— On ne teste donc jamais l'égalité entre deux flottants : on vérifie que leur écart est très petit, par exemple `abs(a - b) < 1e-9`.

Ce qu'il faut savoir faire

— Écrire en binaire un nombre décimal dont l'écriture binaire est finie, comme 0,25 ou 5,75.

7 Représentation des textes

Ce qu'il faut savoir

— L'**ASCII** code 128 caractères sur 7 bits, sans lettres accentuées.

— La norme **ISO-8859-1** (ou Latin-1) code 256 caractères sur un octet, dont les lettres accentuées des langues d'Europe de l'Ouest.

— L'**Unicode** attribue un numéro à chaque caractère de toutes les écritures du monde. Le codage **UTF-8** représente ces numéros sur 1 à 4 octets ; les caractères ASCII y gardent le même code, sur un seul octet.

— Un texte lu avec un autre codage que celui qui a servi à l'écrire affiche des caractères incorrects (« Ã© » au lieu de « é »).

Ce qu'il faut savoir faire

— Utiliser `ord` et `chr` pour passer d'un caractère à son code et inversement.

— Ouvrir un fichier avec le bon codage : `open("f.txt", encoding="utf-8")`.

8 Booléens

Ce qu'il faut savoir

— Un booléen vaut `True` ou `False`. Les opérateurs `not`, `and`, `or` et le ou exclusif (XOR) se décrivent par leur **table de vérité**.

— En Python, `and` et `or` sont évalués de façon **paresseuse** : dans `a and b`, `b` n'est pas évalué si `a` est faux ; dans `a or b`, `b` n'est pas évalué si `a` est vrai. On peut ainsi écrire `i < len(L) and L[i] > 0` sans risquer d'erreur d'indice.

Ce qu'il faut savoir faire

— Dresser la table de vérité d'une expression booléenne.

9 Architecture des ordinateurs

Ce qu'il faut savoir

— Les circuits d'un ordinateur sont construits à partir de **transistors** qui se comportent comme des interrupteurs commandés. Ils réalisent des **portes logiques** (NON, ET, OU, OU exclusif), qu'on combine pour obtenir des circuits plus complexes, comme l'additionneur.

— Dans l'**architecture de von Neumann** (1945), une même mémoire contient les programmes et les données. Le processeur comprend l'**unité arithmétique et logique**, qui fait les calculs, l'**unité de commande**, qui lit et décode les instructions, et des **registres**, petites mémoires très rapides. Des **bus** relient le processeur, la mémoire et les entrées-sorties.

— Le processeur répète un cycle : chercher l'instruction en mémoire, la décoder, l'exécuter.

— La mémoire vive (RAM) perd son contenu à l'arrêt ; la mémoire de masse (disque dur, SSD, clé USB) le conserve.

Ce qu'il faut savoir faire

— Dérouler un programme écrit en langage d'assemblage simple, avec l'aide-mémoire des instructions.

— Construire la table de vérité d'un circuit formé de quelques portes.

10 Web et interactions client-serveur

Ce qu'il faut savoir

— Une page Web est décrite en **HTML** (structure et contenu), mise en forme en **CSS** et rendue interactive en **JavaScript**, langage exécuté par le navigateur.

— Le navigateur (le **client**) envoie des requêtes **HTTP** au **serveur**, qui renvoie les pages. Un formulaire transmet ses données par la méthode **GET**, visible dans l'adresse, ou **POST**, dans le corps de la requête. Le protocole **HTTPS** chiffre les échanges.

— Un programme JavaScript réagit aux **événements** (clic, saisie, survol) et peut modifier la page sans la recharger.

Ce qu'il faut savoir faire

— Écrire une page HTML simple et la mettre en forme avec une feuille de style CSS.

— Distinguer ce qui s'exécute sur le client et ce qui s'exécute sur le serveur.

11 Algorithmique : complexité, correction, tests

Ce qu'il faut savoir

— La **complexité** d'un algorithme mesure l'ordre de grandeur du nombre d'opérations qu'il effectue en fonction de la taille n des données, dans le pire des cas. Un parcours de liste a un coût **linéaire** (de l'ordre de n), deux boucles imbriquées un coût **quadratique** (de l'ordre de n^2).

— Un **variant de boucle** est une quantité entière positive qui diminue strictement à chaque tour : il prouve que la boucle se termine.

— Un **invariant de boucle** est une propriété vraie avant la boucle et qui reste vraie après chaque tour : il prouve que l'algorithme donne le bon résultat.

— La **spécification** d'une fonction décrit ce qu'elle reçoit (préconditions) et ce qu'elle renvoie (postconditions) ; on l'écrit dans la docstring. Des **tests**, écrits avec `assert`, vérifient la fonction sur des exemples, y compris les cas limites (liste vide, un seul élément).

Ce qu'il faut savoir faire

— Évaluer le coût d'un algorithme simple.

— Trouver un variant et un invariant pour une boucle simple.

— Écrire la docstring et les tests d'une fonction.

12 Recherche dichotomique et tris

Ce qu'il faut savoir

— La **recherche dichotomique** cherche une valeur dans une liste **triée** en comparant la valeur à l'élément du milieu, puis en ne gardant que la moitié qui peut la contenir. Son coût est **logarithmique** : environ $\log_2 n$ étapes, soit 20 étapes pour un million d'éléments.

— Le **tri par sélection** place à chaque étape le plus petit élément restant à sa place définitive. Le **tri par insertion** insère chaque élément à sa place parmi les éléments déjà triés. Ces deux tris ont un coût quadratique dans le pire des cas.

Ce qu'il faut savoir faire

— Écrire la recherche dichotomique, le tri par sélection et le tri par insertion.

— Dérouler ces algorithmes à la main sur un exemple.

13 Algorithmes gloutons

Ce qu'il faut savoir

— Un algorithme **glouton** résout un problème d'optimisation en faisant à chaque étape le choix qui semble le meilleur sur le moment, sans jamais revenir sur un choix déjà fait.

— Il est rapide, mais ne donne pas toujours la meilleure solution. Pour le rendu de monnaie avec les pièces et billets en euros, il donne toujours le nombre minimal de pièces ; avec le système de pièces 1, 3 et 4, il rend 6 avec $4 + 1 + 1$ au lieu de $3 + 3$.

Ce qu'il faut savoir faire

— Écrire un algorithme glouton pour le rendu de monnaie ou le problème du sac à dos.

— Trouver un exemple où la solution gloutonne n'est pas optimale.

14 Algorithme des k plus proches voisins

Ce qu'il faut savoir

— L'algorithme des **k plus proches voisins** classe une nouvelle donnée à partir d'un ensemble de données dont la classe est connue : il calcule la distance de la nouvelle donnée à chacune d'elles, garde les k plus proches et attribue la classe majoritaire parmi ces k voisins.

— Le résultat dépend du choix de k et de la distance utilisée (souvent la distance euclidienne).

Ce qu'il faut savoir faire

— Écrire l'algorithme : calcul des distances, tri, vote parmi les k premiers.

— Prévoir la classe donnée par l'algorithme sur un petit exemple, pour différentes valeurs de k.

15 Réseaux et protocole TCP/IP

Ce qu'il faut savoir

— Une **adresse IPv4** est formée de 4 octets, notés en décimal (`192.168.3.1`). La notation `/24` indique que les 24 premiers bits désignent le **réseau**, les 8 derniers la **machine**. Deux machines d'un même réseau local communiquent directement, à travers un **commutateur** ; un **routeur** relie des réseaux différents.

— Les données sont découpées en **paquets**, et chaque couche ajoute ses informations au paquet : c'est l'**encapsulation**.

— Le protocole **TCP** numérote les paquets ; l'expéditeur renvoie un paquet tant que le destinataire n'en a pas accusé réception.

— Le protocole **IP** ajoute au paquet l'adresse de l'expéditeur et celle du destinataire, puis l'achemine de routeur en routeur.

— Le **protocole du bit alterné** illustre la fiabilisation des échanges : chaque message porte un bit, 0 ou 1, qui alterne d'un message au suivant ; l'expéditeur renvoie le message tant qu'il n'a pas reçu l'accusé de réception portant ce même bit.

Ce qu'il faut savoir faire

— Déterminer l'adresse du réseau et le nombre de machines possibles à partir d'une adresse et de son masque.

— Dérouler un échange avec le protocole du bit alterné, y compris en cas de perte d'un message.

16 Systèmes d'exploitation

Ce qu'il faut savoir

— Le **système d'exploitation** gère les ressources de l'ordinateur (processeur, mémoire, fichiers, périphériques) et les partage entre les programmes et les utilisateurs.

— GNU/Linux est un système **libre** : chacun peut l'utiliser, l'étudier, le modifier et le redistribuer. Windows et macOS sont **propriétaires**.

— Les fichiers sont rangés dans une **arborescence** ; un chemin est **absolu** s'il part de la racine `/`, **relatif** s'il part du répertoire courant.

— Chaque fichier a des droits de lecture (`r`), d'écriture (`w`) et d'exécution (`x`) pour son propriétaire (`u`), son groupe (`g`) et les autres utilisateurs (`o`).

Ce qu'il faut savoir faire

— Utiliser les commandes `pwd`, `ls`, `cd`, `mkdir`, `touch`, `cat`, `cp`, `mv`, `rm` et `chmod` (par exemple `chmod u+x script.sh`).

— Écrire le chemin absolu et le chemin relatif d'un fichier dans une arborescence.