TP Chapitre 1 : Affectation, boucle et Si

Objectifs du TP

Objectifs du TP. À l'issue de cette séance, tu dois être capable de :

- reconnaître et utiliser les quatre types de base : `int`, `float`, `bool`, `str` ;
- écrire des affectations et calculer des expressions Python en respectant les priorités ;
- utiliser les opérateurs `//`, `%` et `**` dans des situations de la vie courante ;
- écrire des conditions `if` / `elif` / `else` et des expressions booléennes avec `and`, `or`, `not` ;
- écrire des boucles `for` (avec `range`) et `while`, et éviter les boucles infinies ;
- concevoir et écrire un programme complet à partir d'une spécification (entrées, traitement, sorties) ;
- identifier les erreurs `SyntaxError`, `NameError`, `TypeError`.

Consignes générales. Tous les exercices se traitent en Python (console ou fichier). Enregistre régulièrement ton travail. Quand un programme ne fonctionne pas, lis le message d'erreur : il indique souvent la ligne et le type d'erreur. La partie B se fait *en autonomie* : tu écris des programmes entiers à partir d'une spécification, aucune ligne de code n'est fournie.

Partie A — Application guidée

Les exercices suivants se programment en Python. Ils sont classés du plus simple au plus difficile. Chaque exercice se termine par un *point de validation* (encadré vert) : le résultat exact que doit produire ton programme pour être juste. Complète les squelettes de code en respectant l'indentation.

Exercice A1 — Prix TTC et remise fidélité (instruction `if`)

Un magasin applique la TVA à 20 % sur le prix hors taxes : le prix TTC vaut prix HT × 1,2. Pour tout achat dont le prix TTC dépasse 100 €, le magasin offre une remise fidélité de 10 € sur le total.

Écrire un programme qui demande le prix HT, calcule le prix TTC, applique la remise si nécessaire, puis affiche le prix final.

```
prix_ht = float(input("Prix hors taxes en euros : "))
prix_ttc = prix_ht * 1.2

# A COMPLETER : remise de 10 euros si le prix TTC depasse 100 euros

print("Prix TTC final :", prix_ttc, "euros")
```

Point de validation

Vérifie ton programme : avec 80, il doit afficher `96.0` ; avec 120, il doit afficher `134.0`.

Exercice A2 — Salaire net et niveau de vie (`if` / `elif` / `else`)

Le salaire net mensuel vaut environ 78 % du salaire brut (cotisations sociales). Écrire un programme qui demande le salaire brut, calcule le salaire net, puis affiche un commentaire :

- `Salaire modeste` si le net est inférieur à 1400 € (sous le SMIC net) ;
- `Salaire moyen` si le net est compris entre 1400 € et 2500 € (2500 exclu) ;
- `Salaire confortable` sinon.

```python
brut = float(input("Salaire brut mensuel en euros : "))
net = brut * 0.78

# A COMPLETER : trois cas avec if / elif / else

print("Salaire net :", net, "euros")
```

Point de validation

Vérifie : avec 1500, le programme doit afficher `Salaire modeste` ; avec 2000, `Salaire moyen` ; avec 4000, `Salaire confortable`.

Exercice A3 — Le livret d'épargne (boucles `for` et `while`)

Léa place 200 € sur un livret d'épargne rémunéré à 3 % par an. Chaque année, les intérêts sont ajoutés au capital : le solde est multiplié par 1,03.

1. Écrire un programme qui affiche le solde du livret après chacune des 10 premières années, à l'aide d'une boucle `for`.

```python
solde = 200
for annee in range(1, 11):
    # A COMPLETER : mise a jour du solde puis affichage
    print("Annee", annee, ":", solde, "euros")
```

2. Écrire un second programme qui détermine au bout de combien d'années le capital de Léa dépasse 400 € (le double de son dépôt), à l'aide d'une boucle `while`.

```python
solde = 200
annee = 0
while solde <= 400:     # A COMPLETER : condition
    # A COMPLETER : mise a jour du solde et du compteur d'annees
print("Le capital double apres", annee, "annees.")
```

Point de validation

Premier programme : l'affichage doit commencer par `Annee 1 : environ 206.0 euros` (un léger écart sur les décimales est normal, c'est la représentation des flottants). Second programme : le capital dépasse 400 € après 24 ans.

Exercice A4 — Le distributeur de billets (`//`, `%` et `if`)

Un distributeur automatique ne délivre que des billets de 50 €, 20 € et 10 €. Écrire un programme qui demande un montant (entier multiple de 10) et affiche la décomposition en billets avec *le moins de billets possible* : on prend d'abord le maximum de billets de 50 €, puis de 20 €, puis de 10 €. Si le montant n'est pas un multiple de 10, le programme affiche un message d'erreur.

```python
montant = int(input("Montant a retirer (multiple de 10) : "))
if montant % 10 != 0:
    print("Montant non valide : multiple de 10 attendu.")
else:
    nb50 = montant // 50
    reste = montant % 50
    # A COMPLETER : billets de 20, puis billets de 10
    print("Billets de 50 :", nb50)
    print("Billets de 20 :", nb20)
```

```
10      print("Billets de 10 :", nb10)
```

Point de validation

Vérifie : avec 120, le programme doit afficher 2 billets de 50, 1 billet de 20 et 0 billet de 10 ; avec 190 : 3 billets de 50, 2 billets de 20 et 0 billet de 10.

Partie B — En autonomie

Tu écris maintenant des programmes *entiers*, sans squelette. Pour chaque exercice : lis la spécification (entrées, traitement, sorties, contraintes), réponds d'abord aux questions de conception de l'encadré orange (sur papier ou à voix haute), puis écris ton programme dans un fichier et teste-le avec les valeurs de l'encadré vert. L'objectif est de concevoir toi-même la solution : ne demande pas de lignes de code toutes faites.

Exercice B1 — La caisse du petit commerce (boucle `for` et accumulateur)

Un petit commerce affiche ses prix hors taxes et applique la TVA à 20 % en caisse. Le gérant veut un programme de caisse minimal : on saisit les prix un par un et le programme fait le total.

Énoncé. Écrire un programme qui demande d'abord le nombre d'articles achetés, puis le prix hors taxes de chaque article, et qui affiche le total hors taxes, le montant de la TVA (20 % du total HT), le total TTC et le prix moyen TTC d'un article.

Spécification.

— *Entrées* : un entier N (nombre d'articles, $N \geq 1$), puis N prix hors taxes (flottants, en euros).

— *Traitement* : additionner les prix pour obtenir le total HT ; total TTC = total HT $\times 1{,}2$; TVA = total TTC − total HT ; prix moyen = total TTC $\div N$.

— *Sorties* : le total HT, la TVA, le total TTC et le prix moyen TTC (quatre lignes).

— *Contraintes* : boucle `for` ; une seule variable pour le total (accumulateur) ; les prix ne sont pas mémorisés, ils sont ajoutés au fur et à mesure.

Pour t'aider — questions de conception (sans donner le code)

— Quelle variable joue le rôle d'accumulateur ? Où doit-elle être initialisée (avant ou dans la boucle) et pourquoi ?

— Pourquoi faut-il écrire `float(input(...))` et pas `input(...)` ?

— Que risque-t-il de se passer si $N = 0$? Comment l'éviter ? (réponds en une phrase)

Point de validation

Vérifie ton programme : avec 3 articles à 10 €, 20 € et 30 €, il doit afficher total HT `60.0`, TVA `12.0`, total TTC `72.0` et prix moyen `24.0`.

Exercice B2 — Le relevé de compte (boucle `for` et test du découvert)

Tu veux simuler le relevé d'un compte bancaire : chaque opération fait varier le solde, et le programme doit signaler immédiatement tout passage à découvert.

Énoncé. Écrire un programme qui demande le solde initial, puis le nombre d'opérations, puis pour chaque opération un montant (positif = recette, négatif = dépense). Après chaque opération, il affiche

le nouveau solde ; si le solde est devenu négatif, il affiche en plus un avertissement. À la fin, il affiche le solde final et indique si le compte est à découvert ou non.

Spécification.

— *Entrées* : solde initial (flottant), nombre d'opérations N (entier ≥ 0), puis N montants signés.

— *Traitement* : à chaque opération, solde ← solde + montant ; tester si solde < 0.

— *Sorties* : après chaque opération, `Solde : ... euros` (et `Compte a decouvert !` si solde < 0) ; en fin de programme, le solde final et la synthèse.

— *Contraintes* : boucle `for` ; les opérations ne sont pas mémorisées ; un montant négatif représente une dépense.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi une boucle `for` et pas une boucle `while` ici ?

— Comment distinguer une recette d'une dépense à la saisie ?

— Où placer le test du découvert pour qu'il s'affiche dès que le solde devient négatif ?

— Que doit afficher le programme si $N = 0$?

Point de validation

Vérifie : solde initial 100, opérations −50, −80, +100. Affichages attendus : `Solde : 50.0 euros` ; `Solde : -30.0 euros` puis `Compte a decouvert !` ; `Solde : 70.0 euros` ; puis `Solde final : 70.0 euros`.

Exercice B3 — L'objectif d'épargne (boucle `while` et garde-fou)

Tu épargnes une somme fixe chaque mois pour atteindre un objectif (vacances, ordinateur, permis…). Sans intérêts, on peut calculer exactement au bout de combien de mois l'objectif est atteint.

Énoncé. Écrire un programme qui demande l'apport mensuel (en euros) et l'objectif (en euros), puis simule l'épargne mois par mois : le solde augmente de l'apport à chaque mois. Le programme s'arrête dès que le solde atteint ou dépasse l'objectif, ou au plus tard après 120 mois (10 ans). Il affiche le nombre de mois, le total versé, et un message indiquant si l'objectif a été atteint ou non.

Spécification.

— *Entrées* : apport mensuel (flottant strictement positif), objectif (flottant strictement positif).

— *Traitement* : tant que solde $<$ objectif et mois < 120 : solde ← solde + apport ; mois ← mois + 1.

— *Sorties* : le nombre de mois, le total versé (le solde final), et `Objectif atteint` ou `Objectif non atteint apres 120 mois`.

— *Contraintes* : boucle `while` ; garde-fou de 120 mois obligatoire (terminaison) ; pas d'intérêts dans cet exercice.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi une boucle `while` et pas une boucle `for` ?

— Quelles sont les deux raisons de sortir de la boucle ? Comment le programme fait-il la différence à l'affichage ?

— Pourquoi le garde-fou est-il indispensable ? (pense à un apport très petit, ou nul)

Point de validation

Vérifie : apport 100, objectif 1000 → `Objectif atteint apres 10 mois`, total versé `1000.0` ; apport 50, objectif 3000 → `Objectif atteint apres 60 mois`, total versé `3000.0`.

Exercice B4 — Le distributeur de billets, version « saisie contrôlée » (boucle `while` de saisie)

Le distributeur de la partie A refusait poliment les montants non valides. On veut maintenant qu'il *redemande* une saisie jusqu'à obtenir un montant valide.

Énoncé. Écrire un programme qui demande un montant à retirer. Tant que le montant n'est pas valide — un montant est valide s'il est strictement positif et multiple de 10 — le programme affiche un message d'erreur et redemande. Une fois le montant valide, il décompose le montant en billets de 50 €, 20 € et 10 € avec le moins de billets possible, puis affiche le nombre de billets de chaque sorte.

Spécification.

— *Entrées* : un montant (entier) saisi au clavier, éventuellement plusieurs fois.

— *Traitement* : tant que le montant est invalide (montant ≤ 0 ou montant non multiple de 10), redemander ; puis décomposer : `nb50 = montant // 50`, `reste = montant % 50`, puis `nb20 = reste // 20`, etc.

— *Sorties* : un message d'erreur à chaque saisie invalide ; puis `Billets de 50 : ...`, `Billets de 20 : ...`, `Billets de 10 : ...`.

— *Contraintes* : boucle `while` pour la saisie ; l'opérateur `or` est autorisé ; le montant final est toujours valide.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi la boucle de saisie doit-elle être une boucle `while` ? (combien de saisies au minimum ?)

— Écris en français la condition « le montant est invalide » (deux cas).

— Pourquoi tester la validité avant de décomposer ?

— Que se passerait-il si on oubliait le cas « strictement positif » ? (teste mentalement avec 0)

Point de validation

Vérifie : 120 → 2 billets de 50, 1 de 20, 0 de 10 ; 130 → 2, 1, 1 ; 25 → message d'erreur puis nouvelle saisie ; 0 → message d'erreur puis nouvelle saisie.

Partie C — Exercice de réflexion (à finir à la maison)

À finir à la maison

Mon plan d'épargne : simulateur d'épargne avec objectif.

Contexte. Tu prépares un achat important : ordinateur portable (1000 €), permis de conduire (1800 €), voyage, scooter, etc. Tu décides de placer chaque mois une somme fixe sur un livret d'épargne rémunéré à 3 % par an, les intérêts étant versés une fois par an, en fin d'année.

Objectif. Écrire un programme qui simule l'évolution de ton épargne, année après année, et réponde à des questions comme : au bout de combien de temps mon objectif est-il atteint ? Combien ai-je versé au total ? Combien d'intérêts ai-je gagnés ?

Questions à se poser avant de coder.

— Quelles sont les données d'entrée ? (dépôt initial, apport mensuel, taux annuel, objectif)

— Comment simuler l'épargne mois par mois ? À quel moment faut-il ajouter les intérêts annuels ?

— Quelle boucle utiliser : `for` (on connaît le nombre d'années) ou `while` (on s'arrête quand l'objectif est atteint) ? Pourquoi ?

— Comment garantir que le programme se termine (pas de boucle infinie) ? (tu as déjà vu un garde-fou dans l'exercice B3)

— Comment vérifier que le programme est juste ? (penser à un cas simple : taux 0 %, comme dans l'exercice B3)

Étapes suggérées.

1. Version 0, sans intérêts : calculer le solde après N mois avec une boucle `for` et vérifier le résultat à la main (ex. 100 € par mois pendant 10 mois → 1000 €).
2. Version 1 : ajouter les intérêts annuels de 3 % — attention au moment où ils sont versés (tous les 12 mois).
3. Version 2 : boucle `while` — on s'arrête dès que l'objectif est atteint. Prévoir un garde-fou (nombre maximal d'années, par exemple 50) pour éviter une boucle infinie.
4. Version 3 : afficher un bilan annuel : année, capital, total versé, intérêts cumulés.
5. Bonus : tester plusieurs taux (0 %, 3 %, 4 %) et comparer les durées. Que se passe-t-il si l'apport mensuel augmente de 20 % ?

Contraintes. Données crédibles (taux entre 0 % et 5 %, apport entre 50 € et 300 € par mois, objectif entre 500 € et 5000 €) ; programme testé et commenté ; les messages peuvent être écrits sans accents (ex. `Annee`, `interets`) si ton éditeur pose problème.

Fin du TP — la partie C est à finir à la maison.