TP de synthèse (2h)

Le jeu du pendu

NSI – Première – Bilan du chapitre 1

Ce TP mobilise l'ensemble des notions vues dans ce chapitre : séquences d'instructions, variables, types, structures conditionnelles, boucles bornées et non bornées. Il se déroule en **trois parties, à réaliser dans l'ordre**, sur un unique fichier `.py` que vous complétez au fur et à mesure. À la fin de la séance, vous aurez construit un jeu du pendu fonctionnel.

Partie	Objectif	Durée indicative
A – Comprendre	Lire et tracer des fragments de code	35 min
B – Modifier	Corriger et enrichir une version de base	55 min
C – Créer	Concevoir une extension avec moins de guidage	30 min

Rappel utile : l'opérateur `in` teste si un caractère apparaît dans une chaîne de caractères. Par exemple, `"a" in "chat"` vaut `True`, et `"z" in "chat"` vaut `False`. C'est le seul outil dont vous aurez besoin pour ce TP : aucune notion nouvelle n'est nécessaire.

1 Comprendre

Cette partie présente, séparément, les trois briques qui composeront le jeu du pendu. Vous les assemblerez en partie B.

Exercice 1: L'affichage du mot à trous

On exécute le code suivant :

```
1  mot_secret = "python"
2  lettres_trouvees = "otn"
3
4  affichage = ""
5  for lettre in mot_secret:
6      if lettre in lettres_trouvees:
7          affichage = affichage + lettre
8      else:
9          affichage = affichage + "_"
10
11 print(affichage)
```

1. Tracer l'exécution de la boucle `for` (une ligne par lettre de `mot_secret`), en indiquant à chaque tour si la lettre appartient à `lettres_trouvees`, et la valeur de `affichage` après ce tour.
2. En déduire ce qu'affiche exactement ce programme.
3. Expliquer, avec vos propres mots, à quoi sert chacune des trois variables `mot_secret`, `lettres_trouvees` et `affichage`.
4. Que faudrait-il changer dans ce code pour qu'il fonctionne avec n'importe quel mot secret, et pas seulement `"python"` ?

Exercice 2: Détecter la victoire

On exécute le code suivant, indépendamment du précédent :

```
affichage = "py_ho_"

if "_" in affichage:
    victoire = False
else:
    victoire = True

print(victoire)
```

1. Que va afficher ce programme ? Justifier.
2. Reprendre la question avec `affichage = "python"` à la place de la première ligne.
3. Expliquer le principe utilisé ici pour savoir si le joueur a gagné, à partir de la seule variable `affichage`.

Exercice 3: Une boucle à corriger

On exécute le code suivant :

```
essais_restants = 6
while essais_restants > 0:
    lettre = input("Proposez une lettre : ")
    essais_restants = essais_restants - 1
print("Partie terminée.")
```

1. Combien de fois, au maximum, la boucle `while` tourne-t-elle ?
2. Imaginons qu'on ajoute la logique des deux exercices précédents à l'intérieur de cette boucle (mise à jour de `lettres_trouvees`, recalcul de `affichage`), et qu'un joueur trouve toutes les lettres du mot dès son 2^{e} essai. Que se passerait-il avec ce code tel qu'il est écrit ? Est-ce satisfaisant ?
3. D'après ce que vous avez vu à l'exercice 2, quelle condition supplémentaire faudrait-il ajouter à la condition de la boucle `while`, en plus de `essais_restants > 0`, pour que la boucle s'arrête aussi dès que le mot est entièrement trouvé ?

2 Modifier

Voici une version de base du jeu du pendu, qui assemble les trois briques de la partie précédente – mais qui contient exactement le défaut identifié à l'exercice 3.

Exercice 4: Version de base (fournie)

```
mot_secret = "python"
lettres_trouvees = ""
essais_restants = 6

while essais_restants > 0:
    affichage = ""
```

```
    for lettre in mot_secret:
        if lettre in lettres_trouvees:
            affichage = affichage + lettre
        else:
            affichage = affichage + "_"
    print(affichage)

    proposition = input("Proposez une lettre : ")

    if proposition in mot_secret:
        lettres_trouvees = lettres_trouvees + proposition
    else:
        essais_restants = essais_restants - 1

print("Partie terminée.")
```

Copier ce code dans votre fichier `.py`, l'exécuter, et vérifier par vous-même le défaut décrit à l'exercice 3 : jouez une partie en trouvant toutes les lettres correctement, et observez ce qu'il se passe une fois le mot entièrement découvert.
Puis, apporter les modifications suivantes, **dans l'ordre** :

1. Corriger la condition de la boucle `while` pour qu'elle s'arrête aussi dès que le mot est entièrement trouvé (utiliser le résultat de l'exercice 3). Attention : la variable `affichage` doit exister **avant** le premier test de la condition ; il faudra donc l'initialiser avant la boucle (par exemple avec des `_` partout, puisqu'aucune lettre n'est encore trouvée).
2. Ajouter, après la boucle, un message qui indique si le joueur a **gagné** (le mot est entièrement trouvé) ou **perdu** (il ne reste plus d'essais). En cas de défaite, afficher également le mot secret.
3. Ajouter une variable `essais_utilises` qui compte le nombre de propositions **incorrectes** faites par le joueur, et l'afficher dans le message de victoire (par exemple : « Bravo, trouvé avec 2 erreur(s) ! »).

Indication

Pour la question 1 : initialisez `affichage` avec autant de `_` que de lettres dans `mot_secret`, à l'aide d'une petite boucle `for` placée avant le `while` (une boucle très semblable à celle qui reconstruit `affichage` à l'intérieur du `while`, mais qui se contente d'ajouter des `_` sans jamais tester `lettre in lettres_trouvees`, puisqu'aucune lettre n'est encore trouvée).

3 Créer

Cette partie est plus ouverte : vous devez concevoir vous-même la solution, sans code de départ.

Exercice 5: Enchaîner plusieurs mots

Reprendre votre programme de la partie B, dans un nouveau fichier nommé `pendu_final.py` et le transformer pour que le joueur enchaîne **3 mots secrets** à la suite (par exemple `"python"`, `"variable"`, `"boucle"`, mais vous pouvez choisir d'autres mots). Le joueur gagne un point à chaque mot trouvé (peu importe le nombre d'essais

utilisés). À la fin des 3 mots, afficher le score final, sous la forme « Score final : X / 3 ».
Contrainte : vous n'avez pas encore vu les listes en cours. Utilisez uniquement ce que vous connaissez déjà (variables, types, conditionnelles, boucles `for` et `while`) pour résoudre ce problème.

Indication

Le nombre de mots à enchaîner (3) est connu à l'avance : quelle boucle, vue dans ce chapitre, est adaptée pour répéter une action un nombre de fois connu à l'avance ? À l'intérieur de cette boucle, vous connaissez le numéro du mot en cours (1, 2 ou 3) : une structure conditionnelle permet de choisir quel mot secret utiliser selon ce numéro, sans avoir besoin d'une liste.

Exercice 6: Une amélioration de votre choix

Proposer et coder **une** amélioration supplémentaire de votre choix pour ce jeu du pendu. En voici quelques idées, mais vous pouvez en imaginer d'autres :

- accepter aussi bien les majuscules que les minuscules (par exemple, « P » et « p » doivent être acceptées comme la même lettre) ;
- empêcher qu'une lettre déjà proposée ne soit comptée une seconde fois comme une erreur si le joueur la retape par mégarde ;
- afficher, en plus du mot à trous, le nombre d'essais qu'il reste au joueur à chaque tour.

Dans tous les cas, expliquer en une phrase ce que votre amélioration apporte au jeu.

Remarque : Ce qu'il faut retenir de ce TP

Un programme, même modeste comme ce jeu du pendu, s'obtient rarement d'un seul jet : on part d'un fragment qu'on comprend, on l'assemble avec d'autres, on corrige les défauts qu'on observe en le testant, puis on l'enrichit progressivement. C'est exactement la démarche que vous avez suivie dans ce TP, et c'est celle que vous emploierez pour tous les programmes plus conséquents que vous écrirez cette année.