Guide des programmes Python convertis depuis Scratch

1. Question-réponse (2+2=?)

Description du programme

Ce programme simule une interaction de type Scratch où un personnage (le lutin) pose une question mathématique simple et évalue la réponse de l'utilisateur.

Fonctionnement

- Le programme pose la question : "2+2=?"
- L'utilisateur tape sa réponse
- Le programme compare la réponse à "4"
- Si correct : affiche "Exact" pendant 5 secondes
- Si incorrect : affiche "Faux" pendant 5 secondes

Fonction personnalisée

Le programme définit une fonction **`dire(message, duree)`** qui simule le comportement du bloc Scratch "dire [...] pendant (n) secondes". Cette fonction :

- Affiche le message avec print()
- Attend pendant la durée spécifiée avec time.sleep()

Comparaison de chaînes

Important : le programme compare la réponse comme une **chaîne de caractères** (texte), pas comme un nombre. Donc la réponse "4" (texte) est comparée à "4" (texte). Si l'utilisateur tape "4.0" ou " 4 " (avec espace), ce sera considéré comme faux.

Équivalence Scratch-Python

Bloc Scratch	Équivalent Python
demander [2+2=?] et attendre	`reponse = input("2+2=?")`
réponse = 4	`reponse = "4"`

2. Tirages aléatoires

Description du programme

Ces deux programmes sont très similaires. Ils tirent au hasard deux nombres (A et B) et comptent combien de tirages sont nécessaires pour que A et B soient égaux.

Deux versions :

- **Version 1 à 1000** : Tire des nombres entre 1 et 1000
- **Version 1 à 10000** : Tire des nombres entre 1 et 10000

Fonctionnement

Le programme utilise une boucle **`while`** qui continue tant que A ≠ B. À chaque itération :

- On tire un nouveau A au hasard
- On tire un nouveau B au hasard
- On incrémente le compteur C
- On vérifie si A == B pour sortir de la boucle

Variables utilisées

Variable	Description
`A`	Premier nombre tiré au hasard
`B`	Deuxième nombre tiré au hasard
`C`	Compteur du nombre de tirages effectués

Probabilité

La probabilité que deux nombres soient égaux est de 1/N où N est la plage de valeurs. En moyenne :

- **Version 1-1000** : environ 1000 tirages nécessaires
- **Version 1-10000** : environ 10000 tirages nécessaires

3. Jeu : Trouver le nombre

Description du programme

Ce programme est un jeu de devinette. L'ordinateur choisit un nombre secret entre 1 et 1000, et le joueur doit le deviner. À chaque tentative, le programme indique si le nombre proposé est trop petit, trop grand, ou correct.

Déroulement du jeu

- L'ordinateur tire un nombre secret B entre 1 et 1000
- Le joueur propose un nombre A
- Le programme compare A et B et affiche un indice
- Le compteur C est incrémenté
- La boucle continue jusqu'à ce que A == B

Structure conditionnelle

Le programme utilise une structure **`if/elif/else`** pour donner des indices :

Condition	Message affiché
`A == B`	"Gagné !"
`A < B`	"Trop petit !"
`A > B`	"Trop grand !"

Stratégie optimale

La meilleure stratégie pour gagner rapidement est la **recherche binaire** : on propose toujours le milieu de l'intervalle possible. Cela permet de trouver le nombre en **maximum 10 tentatives**.

4. Calcul de remise

Description du programme

Ce programme calcule une remise commerciale en fonction du montant d'une commande. Le taux de remise est de 1% pour les commandes inférieures à 30000 euros, et de 2% pour les commandes de 30000 euros ou plus.

Algorithme

- Demander le montant M de la commande
- Si M < 30000 : appliquer un taux de 1%
- Sinon : appliquer un taux de 2%
- Calculer la remise R = M × (taux / 100)
- Afficher le taux et le montant de la remise

Structure conditionnelle

Le programme utilise une structure **`if/else`** pour déterminer le taux de remise applicable.

Montant commande	Taux	Exemple de remise
M < 30 000 €	**1%**	10 000 € → 100 € de remise
M ≥ 30 000 €	**2%**	50 000 € → 1 000 € de remise

Variables utilisées

- **`M`** : Montant de la commande (saisi par l'utilisateur)
- **`taux`** : Taux de remise applicable (1 ou 2)
- **`R`** : Montant de la remise calculée

5. Jeu Pac-Man

Description du programme

Ce programme est un jeu Pac-Man complet avec interface graphique. Le joueur contrôle Pac-Man dans un labyrinthe et doit manger tous les points tout en évitant les fantômes.

Bibliothèques utilisées

- **tkinter** : Interface graphique et canvas pour dessiner le jeu
- **random** : Génération de mouvements aléatoires pour certains fantômes
- **time** : Gestion du temps (non utilisé directement dans la version finale)

Éléments du jeu

Élément	Description
Labyrinthe	Grille 19×21 avec murs (1), chemins (0), points (2) et super-points (3)
Pac-Man	Personnage jaune contrôlé par le joueur avec animation de bouche
Fantômes	8 fantômes avec 4 comportements différents (poursuite, patrouille, embuscade, aléatoire)
Points	Petits points (10 points) et super-points (50 points + mode power)

Fonctions principales

__init__(self, root) : Initialise le jeu, crée le canvas, définit le labyrinthe et les personnages

move_pacman(self) : Déplace Pac-Man selon la direction choisie, gère la collision avec les points

move_ghosts(self) : Déplace les fantômes selon leur comportement individuel (IA)

check_collisions(self) : Vérifie les collisions entre Pac-Man et les fantômes

draw(self) : Dessine tous les éléments du jeu (labyrinthe, Pac-Man, fantômes, score)

game_loop(self) : Boucle principale qui met à jour le jeu toutes les 120 millisecondes

Règles du jeu

- Utilisez les flèches directionnelles pour déplacer Pac-Man
- Mangez tous les points pour gagner
- Évitez les fantômes (ils vous font perdre une vie)
- Les super-points activent le mode power : vous pouvez manger les fantômes !
- Vous avez 3 vies au total

Conclusion

Ces cinq programmes illustrent différents concepts fondamentaux de la programmation :

- **Boucles** : while pour répéter des actions
- **Conditions** : if/elif/else pour prendre des décisions
- **Variables** : pour stocker et manipuler des données
- **Fonctions** : pour organiser et réutiliser du code
- **Aléatoire** : random.randint() pour générer des nombres au hasard
- **Interaction utilisateur** : input() pour demander des informations

Tous ces programmes ont été convertis depuis des projets Scratch (.sb3) vers Python, démontrant la transition naturelle entre la programmation visuelle et la programmation textuelle.

- **Scratch** est un langage **graphique** idéal pour débuter dès 6 ans : on assemble des blocs pour créer des jeux, sans se soucier des erreurs de code.
- **Python** est un langage **textuel** utilisé par les professionnels, qui demande d'écrire des instructions précises avec un clavier.
- **Scratch** se concentre sur l'apprentissage de la **logique** (boucles, conditions) de manière ludique et visuelle.
- **Python** est un véritable outil pour construire des projets concrets (sites web, IA, sciences) et ouvre des portes professionnelles.
- **En bref**, Scratch est une excellente porte d'entrée pour les enfants, tandis que Python est la destination idéale pour aller plus loin, dès l'adolescence ou l'âge adulte.