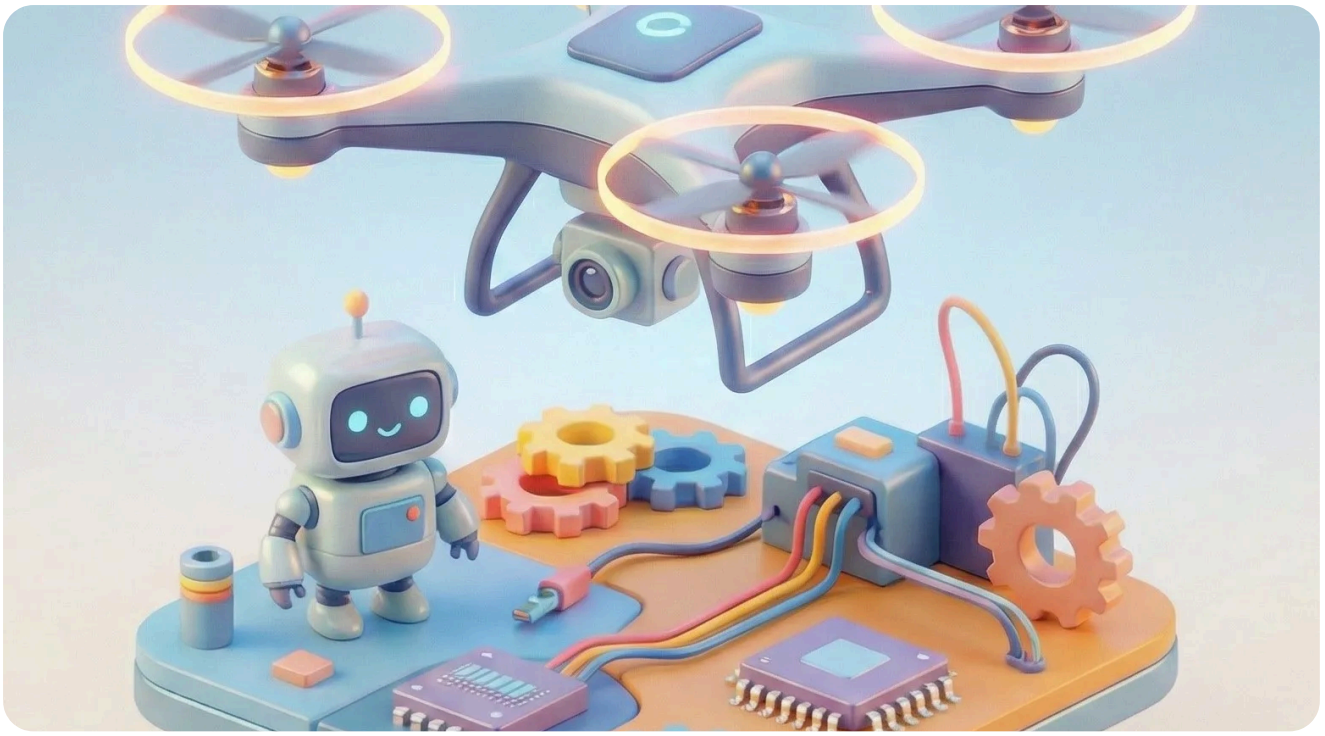


Drones

Drones

Thibault Six · Théo Vandormael

29 chapitres

SUPPORTS DE COURS

https://cours.tsix.be/cours/drones

Version 1 · 12/09/2026

PRÉSENTATION

Cours de Drones et Robotique

Bienvenue dans ce cours qui vous emmènera de vos premières lignes de code jusqu'au pilotage et à la programmation de drones autonomes !

Ce que vous allez apprendre

Ce parcours combine quatre disciplines complémentaires : la programmation, la robotique, l'électronique et les drones. Chaque étape prépare la suivante, en construisant progressivement vos compétences jusqu'à maîtriser des systèmes complexes.

Vous commencerez par découvrir la logique algorithmique et les bases de la programmation à travers des environnements visuels et des robots Lego. Ensuite, vous plongerez dans l'électronique avec Arduino et Raspberry Pi pour construire vos propres circuits et robots. Enfin, vous appliquerez toutes ces connaissances au monde des drones : réglementation, pilotage, montage et programmation.

Structure du cours

Le cours se divise en trois grandes parties : une introduction à la programmation et à la robotique pour poser les bases, une exploration approfondie de l'électronique qui constitue le cœur technique du parcours, et enfin l'application concrète aux drones.

Contacter le professeur

Par email : thibault.six@ifosup.wavre.be

Prêt à décoller ? Consultez l'introduction pour découvrir le programme détaillé.

Sommaire

- Cours de Drones et Robotique

1 Introduction
- 1.1 Évaluations
- 1.2 Algo-bot
- 1.3 Liste d'achats
- 1.4 Programmation
- 1.5 Scratch
- 1.6 Variables et Calculs
- 1.7 Plan cartésien
- 1.8 Formes géométriques

2 Lego
- 2.1 Premiers Pas avec le Robot EV3
- 2.2 Annexe : Le Paramètre Direction
- 2.3 EV3 : Couleurs, Angles et Intégration

3 Processing
- 3.1 Outils de dessin
- 3.2 Couleurs
- 3.3 Premières formes
- 3.4 Variables
- 3.5 Conditions
- 3.6 Boucles

4 Arduino
- 4.1 IDE Arduino
- 4.2 Potentiomètre et valeurs analogiques
- 4.3 Le capteur de température (DHT11)

5 Drones
- 5.1 Sinus et cosinus
- 5.2 Comment ça vole ?
- 5.3 Premiers Pas avec Python
- 5.4 Pilote ton Drone Tello avec Python

CHAPITRE 1

Introduction au cours

Slides du chapitre

- intro-drones.pdf

Objectifs pédagogiques

À l'issue de ce cours, vous serez capable de comprendre et appliquer les concepts fondamentaux de la programmation (variables, conditions, boucles, fonctions), de concevoir et réaliser des montages électroniques avec Arduino et Raspberry Pi, de construire des systèmes autonomes (station météo, robots roulants) et enfin de piloter, monter et programmer des drones en respectant la réglementation en vigueur.

Une approche collaborative

Ce cours privilégie l'apprentissage par la pratique et l'expérimentation. Chaque concept théorique est immédiatement mis en application à travers des projets concrets. Les idées de projets, les questions et les découvertes de chacun enrichissent le parcours collectif. N'hésitez pas à proposer vos propres expérimentations !

Programme détaillé

Partie 1 — Programmation et robotique

Cette première partie pose les fondations. Nous y découvrons la logique algorithmique et les principes de base de la programmation à travers trois environnements progressifs.

Algobot — Un serious game belge qui introduit les concepts d'instructions, de fonctions et de boucles en guidant un robot à travers des puzzles. Une entrée en matière ludique et accessible.

Scratch — L'environnement de programmation visuelle par blocs permet d'approfondir les notions de variables, conditions et événements sans se soucier de la syntaxe d'un langage.

Lego Mindstorm — Passage au monde physique avec la construction et la programmation de robots. L'interface reste proche de Scratch, mais les défis du monde réel apparaissent : calibration des capteurs, gestion des moteurs, obstacles imprévus.

Partie 2 — Électronique

Cette partie constitue le cœur technique du cours. Nous y construisons des systèmes de plus en plus complexes.

Arduino — Introduction aux cartes de développement et à la programmation embarquée en C++. Nous connectons des capteurs (température, distance, lumière), des moteurs et divers composants pour comprendre les fondements de l'électronique.

Raspberry Pi — Plus puissant qu'un Arduino, ce mini-ordinateur se programme en Python. Il ouvre la porte à des projets nécessitant davantage de ressources de calcul.

Montages électroniques — LEDs, moteurs, écrans : nous assemblons des circuits progressivement plus élaborés. Chaque montage renforce la compréhension des principes électroniques.

Station météo — Premier projet complet intégrant plusieurs capteurs (température, humidité, pression) et l'affichage des données. Un excellent exercice d'intégration avant les systèmes mobiles.

Robots roulants — Point culminant de cette section : des robots autonomes capables de se déplacer, éviter les obstacles et interagir avec leur environnement. La préparation idéale pour les drones.

Partie 3 — Drones

Toutes les compétences acquises convergent vers cette dernière partie.

Permis théorique — La réglementation belge et européenne encadre strictement l'usage des drones. Nous étudions les règles essentielles pour voler en toute légalité.

Pilotage — Avant de programmer un drone, il faut savoir le contrôler manuellement. Entraînement sur simulateurs puis sur appareils réels.

Montage — Comprendre l'anatomie d'un drone : châssis, moteurs, contrôleur de vol, batterie, hélices. Savoir diagnostiquer et réparer.

Programmation Python — Avec les drones DJI Tello, nous programmons des vols automatisés, des patrouilles et des missions complexes.

Matériel

À l'école

L'école dispose de tout le matériel nécessaire : kits Arduino et Raspberry Pi, composants électroniques variés (LEDs, moteurs, capteurs, écrans), et drones DJI Tello pour l'apprentissage.

Nous ne disposons pas de drones perfectionnés type DJI Mavic. Pour explorer ces appareils, un achat personnel serait nécessaire.

Kits personnels (optionnel)

Pour pratiquer à domicile ou approfondir les projets :

Kit Arduino — Des starter kits complets sont disponibles entre 25€ et 100€. Le kit Elegoo Super Starter Kit est recommandé pour débuter : lien Amazon.

Multimètre — Outil indispensable pour diagnostiquer les circuits, environ 40€ : lien Amazon.

Drone personnel — Les DJI Mavic (à partir du Mini 3) offrent d'excellentes performances. Prix de 400€ à plus de 2000€ : DJI Store.

Prochaine étape : découvrir Algobot, notre premier outil d'apprentissage.

1.1 Chapitre : Évaluations et Dossiers Pédagogiques

> 🎯 ***Ce qu'il faut retenir** : Ce cours est sanctionné par 3 unités d'enseignement distinctes. Chaque évaluation est binaire : **Réussi** ou **Non réussi**. Pas de cote sur 20.*

Slides du chapitre

- évaluations.pdf

🎓 Les 3 Unités d'Enseignement

Ce cours couvre trois dossiers pédagogiques officiels. Chacun mène à un **certificat** distinct.

#	Unité d'Enseignement	Heures	Type de certificat
UE1	Introduction à l'informatique	20h	Secondaire supérieur de transition
UE2	Initiation à la programmation	40h	Enseignement supérieur (adultes, type court)
UE3	Sensibilisation au télépilotage drone	24h	Secondaire supérieur de transition

📋 Détail des Évaluations

UE1 : Introduction à l'informatique

Pas d'examen formel. Évaluation continue basée sur la participation aux activités.

Ce qu'on attend de toi :

- Mettre en route un système informatique et clôturer une session
- Gérer des fichiers et répertoires (créer, déplacer, supprimer)
- Créer et imprimer un document

💡 *Tu valides cette UE en participant activement aux séances.*

UE2 : Initiation à la programmation

Évaluation : Projet personnel

Tu devras concevoir et réaliser un **montage électronique original** avec Arduino :

- Choix des capteurs et actionneurs
- Câblage du circuit
- Programmation en C++ (structure, variables, conditions, boucles)

Critères d'évaluation :

- Démarche de résolution cohérente
- Représentation du programme (pseudocode, schéma)
- Code fonctionnel qui répond au problème posé

💡 *Pendant le cours, on construira progressivement des montages de plus en plus complexes, jusqu'à un "drone roulant" Arduino.*

UE3 : Sensibilisation au télépilotage

Évaluation : Épreuve pratique de pilotage

Tu devras réussir un **parcours de vol** en pilotage manuel avec le drone Tello :

- Démontrer ta capacité à télépiloter
- Respecter les consignes de sécurité
- Connaître les mesures de protection (pour toi et les tiers)

Pendant le cours :

- Théorie : réglementation, droit à l'image
- Pratique : montage/démontage, exercices de pilotage progressifs

💡 *Le parcours final sera annoncé en fin de formation. Pas de surprise : tu t'entraîneras sur des exercices similaires.*

✅ Récapitulatif

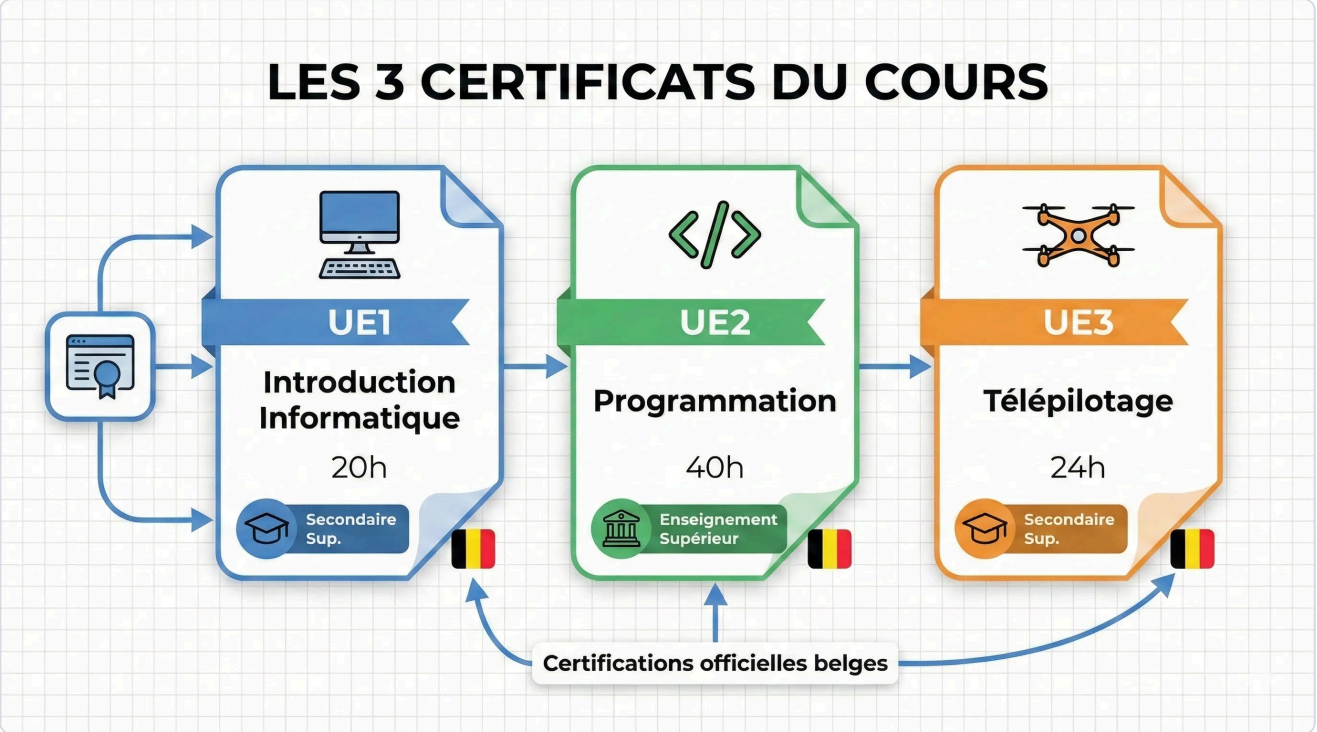
UE	Forme d'évaluation	Quand ?
UE1	Participation continue	Tout au long du cours
UE2	Projet Arduino (montage + code)	Fin du module programmation
UE3	Parcours drone (pilotage manuel)	Dernière(s) séance(s)

🏆 Ce que tu obtiens

À la réussite des 3 UE, tu reçois :

- **2 certificats** de l'enseignement secondaire supérieur de transition (UE1 + UE3)
- **1 certificat** de l'enseignement supérieur pour adultes (UE2)

Ces certifications sont officielles et reconnues en Belgique.



❓ Questions Fréquentes

"Je peux rater une UE et quand même réussir les autres ?" → Oui, chaque UE est indépendante.

"Y a-t-il un examen écrit ?" → Non. L'évaluation est pratique et basée sur des projets/démonstrations.

"Je peux repasser si je rate ?" → Oui, tu as droit à une deuxième session.

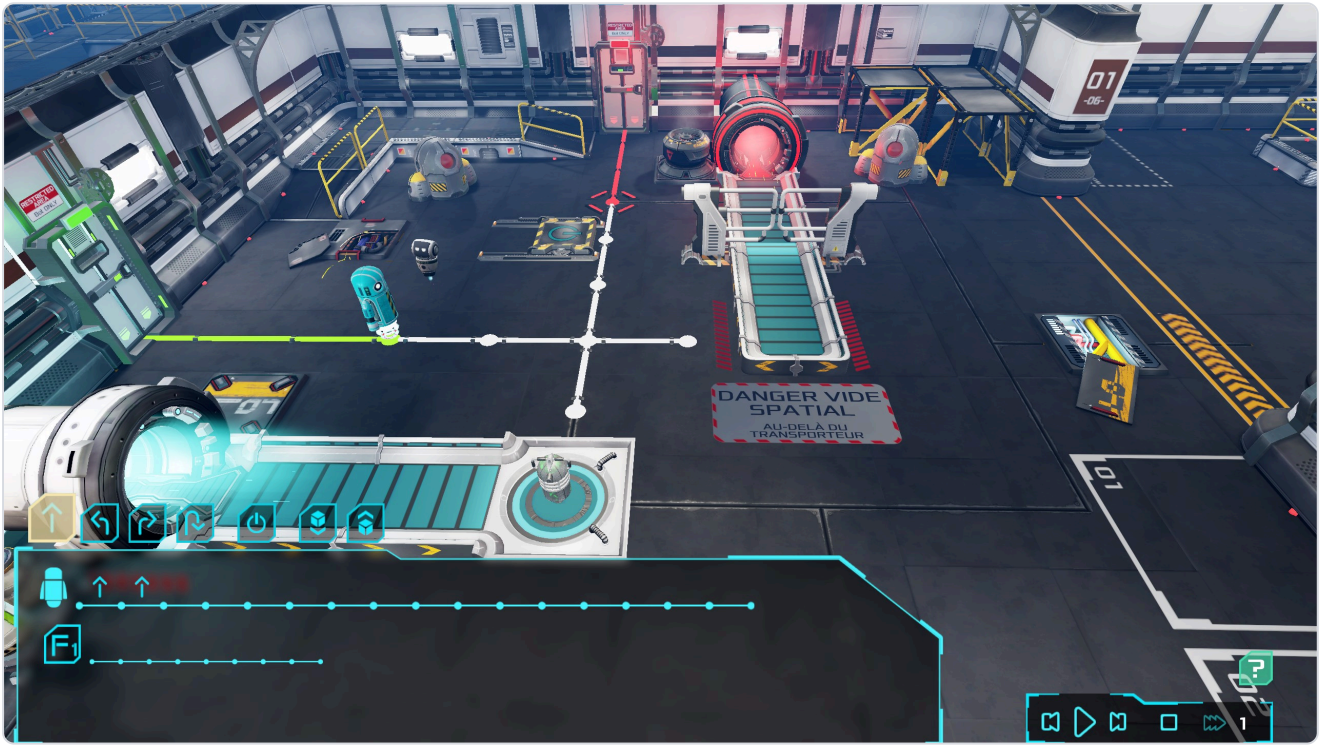
1.2 Algo-bot

Slides du chapitre

- algo-bot.pdf

Un serious game belge primé

Algobot est un jeu vidéo éducatif (serious game) développé en Belgique par le centre de compétence numérique Technobel, basé à Ciney, en collaboration avec le studio de développement Fishing Cactus. Lancé en 2013, il a rapidement conquis la scène internationale du serious gaming.



Palmarès

Dès sa sortie, Algobot s'est distingué lors de plusieurs événements prestigieux. Il a été présenté à la conférence EDEN à Oslo devant plus de 400 spécialistes de l'e-learning. En novembre 2013, il remporte le trophée du "Meilleur Serious Game" dans la catégorie RH-Formation au Serious Game Expo de Lyon. En juillet 2014, il décroche la deuxième place aux European Serious Game Awards à Bucarest, face à des concurrents développés par de grandes universités européennes.

Ces reconnaissances internationales témoignent de la qualité pédagogique du jeu et de l'expertise belge en matière de formation numérique innovante.



L'histoire du jeu

Sauvez l'Europa !

Algobot est un petit robot maladroit au service de l'équipage de l'Europa, un vaisseau intergalactique transportant une partie de l'humanité ayant fui la Terre devenue inhabitable. Les passagers hibernent dans l'attente d'un nouveau monde, tandis qu'une équipe de robots assure le fonctionnement du vaisseau.

Tout se déroule normalement jusqu'à ce qu'une mission de routine tourne mal et sème la panique à bord. Le désordre est tel que seuls Algobot et son supérieur PAL peuvent sauver le vaisseau d'une fin catastrophique. Pour y parvenir, ils doivent rétablir l'Intelligence Artificielle du vaisseau, sans laquelle l'équipage humain ne pourrait jamais se réveiller.

C'est là que vous intervenez : en prenant le contrôle d'Algobot à travers des lignes de commandes, vous résolvez des énigmes pour déverrouiller des portes, débloquer des passages et finalement réactiver l'IA vitale.

Concepts abordés

Derrière ce scénario de science-fiction se cachent les principes fondamentaux de la programmation. En jouant, vous manipulez sans vous en rendre compte des concepts que vous retrouverez dans tous les langages de programmation.

Les instructions

Chaque commande que vous donnez à Algobot est une instruction : avancer, tourner, interagir avec un objet. Un programme n'est qu'une suite d'instructions exécutées dans l'ordre.

Les boucles

Pourquoi écrire dix fois "avancer" quand on peut dire "répéter 10 fois : avancer" ? Les boucles permettent d'exécuter plusieurs fois un même bloc d'instructions.

Les fonctions

Quand une séquence d'instructions revient souvent, on peut lui donner un nom et la réutiliser. Les fonctions rendent le code plus lisible et évitent les répétitions.

Les variables

Stocker une valeur pour la réutiliser plus tard : c'est le rôle des variables. Elles permettent à votre programme de "mémoriser" des informations.

Les conditions

Prendre des décisions selon le contexte : "si la porte est fermée, alors appuyer sur le bouton". Les conditions permettent à votre programme de s'adapter aux situations.

Pourquoi commencer par Algobot ?

Ces concepts (instructions, boucles, fonctions, variables, conditions) sont universels. Vous les retrouverez dans Scratch, Python, C++, et dans la programmation de drones. En les découvrant dans un environnement ludique et sans syntaxe complexe, vous construisez des fondations solides.

Algobot développe aussi la pensée algorithmique : décomposer un problème en étapes simples, anticiper le comportement du robot, identifier et corriger les erreurs quand le résultat n'est pas celui attendu.

Téléchargement et installation

Algobot est gratuit et disponible sur plusieurs plateformes.

Site officiel

https://www.algobot.be/

Version ordinateur

Téléchargez la version correspondant à votre système d'exploitation :

- Windows 32 bits
- Windows 64 bits
- Mac
- Linux

Une fois téléchargé, décompressez l'archive et lancez l'exécutable. Aucune installation n'est nécessaire.

Version mobile (tablette)

Algobot est également disponible sur tablette pour jouer du bout des doigts :

- App Store (iOS) — Tablettes de 2017 ou plus récentes
- Google Play (Android) — Tablettes de 2018 ou plus récentes avec une résolution minimum de 1366×768 pixels

Note : L'application mobile nécessite une connexion depuis la Belgique et un store configuré sur "Belgique" pour être installable.

Une fois Algobot installé, lancez-vous dans les premiers niveaux pour vous familiariser avec l'interface. Nous passerons ensuite à Scratch pour approfondir ces concepts.

1.3 Liste d'achats Arduino

- arduino list achats.pdf

1.4 Introduction à la Programmation

> 🎯 ***Objectif*** *: Comprendre ce qu'est la programmation, à quoi elle sert, et découvrir les 4 concepts fondamentaux qu'on retrouve dans TOUS les langages de programmation.*

Slides du cours

- introduction programmation.pdf

1. C'est quoi, la programmation ?

Définition simple

Programmer, c'est donner des instructions à une machine pour qu'elle exécute une tâche.

Une machine (ordinateur, robot, drone, smartphone...) est fondamentalement "bête" : elle ne sait rien faire toute seule. Elle a besoin qu'on lui explique EXACTEMENT ce qu'elle doit faire, étape par étape, dans le bon ordre.

Un programme, c'est donc une **liste d'instructions** écrites dans un langage que la machine peut comprendre.

Analogie : La recette de cuisine 🍳

Imagine que tu dois expliquer à quelqu'un qui n'a JAMAIS cuisiné comment faire une omelette. Tu ne peux pas juste dire "fais une omelette" — il faut détailler :

1. Prends 3 œufs
2. Casse-les dans un bol
3. Bats-les avec une fourchette
4. Fais chauffer une poêle avec du beurre
5. Verse les œufs battus dans la poêle
6. Attends 2 minutes
7. Retourne l'omelette
8. Attends 1 minute
9. Sers dans une assiette

C'est exactement ce que fait un programmeur : il écrit une "recette" ultra-détaillée pour la machine.

2. La programmation est PARTOUT

Tu utilises des programmes tous les jours sans t'en rendre compte :

Objet	Programme qui tourne dedans
📱 Smartphone	Applications, système d'exploitation
🚗 Voiture	Système de freinage ABS, GPS, régulateur
🚦 Feu rouge	Minuterie et capteurs de circulation
🏧 Distributeur de billets	Vérification de carte, calcul du solde
🎮 Console de jeux	Jeux vidéo, menus

Objet	Programme qui tourne dedans
🤖 Robot aspirateur	Navigation, détection d'obstacles
✈️ Avion	Pilote automatique, instruments de bord

Et dans ce cours ? On va programmer :

- Des **robots** (Lego Mindstorm, robots roulants Arduino)
- Des **montages électroniques** (LEDs, capteurs, écrans)
- Des **drones** (Tello avec Python)

3. Les 4 concepts fondamentaux

Que tu programmes en Scratch, Python, C++, JavaScript ou n'importe quel autre langage, tu retrouveras TOUJOURS ces 4 concepts de base. Apprends-les une fois, utilise-les partout !

3.1 Les Instructions

Une **instruction** est un ordre simple donné à la machine.

Exemples :

- "Avance de 10 pas"
- "Allume la LED"
- "Affiche 'Bonjour' à l'écran"
- "Joue un son"

Un programme est une **séquence d'instructions** exécutées dans l'ordre, de haut en bas.

⚠️ **L'ordre est crucial !** Si tu mets tes chaussettes APRÈS tes chaussures, ça ne marche pas...

3.2 Les Variables

Une **variable** est comme une boîte étiquetée dans laquelle on peut stocker une valeur.

```
┌──────────────┐
│    score     │  ← L'étiquette (le nom)
├──────────────┤
│      42      │  ← Le contenu (la valeur)
└──────────────┘
```

Caractéristiques importantes :

- La boîte a un **nom** (l'étiquette) : `score`, `vie`, `nom_joueur`...

- La boîte contient une **valeur** : un nombre, du texte, vrai/faux...
- On peut **lire** ce qu'il y a dans la boîte
- On peut **modifier** le contenu (la nouvelle valeur REMPLACE l'ancienne)

Exemple concret — Le score dans un jeu :

```
Au début :          score = 0
Étoile attrapée :   score = score + 10   → score vaut maintenant 10
Autre étoile :      score = score + 10   → score vaut maintenant 20
```

⚠️ **Piège classique :** Une variable ne peut contenir qu'UNE SEULE valeur à la fois. Si tu fais `x = 5` puis `x = 7`, la variable `x` contient 7 (pas 12, pas les deux !).

3.3 Les Conditions

Une **condition** permet à la machine de prendre des décisions : faire quelque chose SI une situation est vraie.

Structure de base :

```
SI (quelque chose est vrai) ALORS
    faire ceci
SINON
    faire cela
```

Exemple — Le feu rouge :

```
SI le feu est rouge ALORS
    arrête-toi
SINON SI le feu est orange ALORS
    ralentis
SINON
    avance
```

Exemple — Jeu vidéo :

```
SI vies = 0 ALORS
    afficher "Game Over"
SINON
    continuer le jeu
```

On peut aussi **combiner** des conditions avec ET / OU :

- `SI il pleut ET j'ai pas de parapluie ALORS je reste à la maison`
- `SI c'est samedi OU c'est dimanche ALORS c'est le weekend`

3.4 Les Boucles

Une **boucle** permet de répéter des instructions plusieurs fois sans les réécrire.

Sans boucle (pénible) :

```
avancer
avancer
avancer
avancer
avancer
```

Avec boucle (malin) :

```css
RÉPÉTER 5 FOIS
    avancer
```

Il existe 2 grands types de boucles :

Type	Quand l'utiliser	Exemple de la vie réelle
RÉPÉTER X FOIS	Tu sais exactement combien de fois	"Fais 20 pompes" 💪
TANT QUE	Tu ne sais pas combien de fois, mais tu connais la condition d'arrêt	"Mange tant qu'il reste des frites" 🍟

Exemple — Robot qui marche :

```css
RÉPÉTER 10 FOIS
    lever jambe droite
    avancer jambe droite
    poser jambe droite
    lever jambe gauche
    avancer jambe gauche
    poser jambe gauche
```

Exemple — Lecteur de musique :

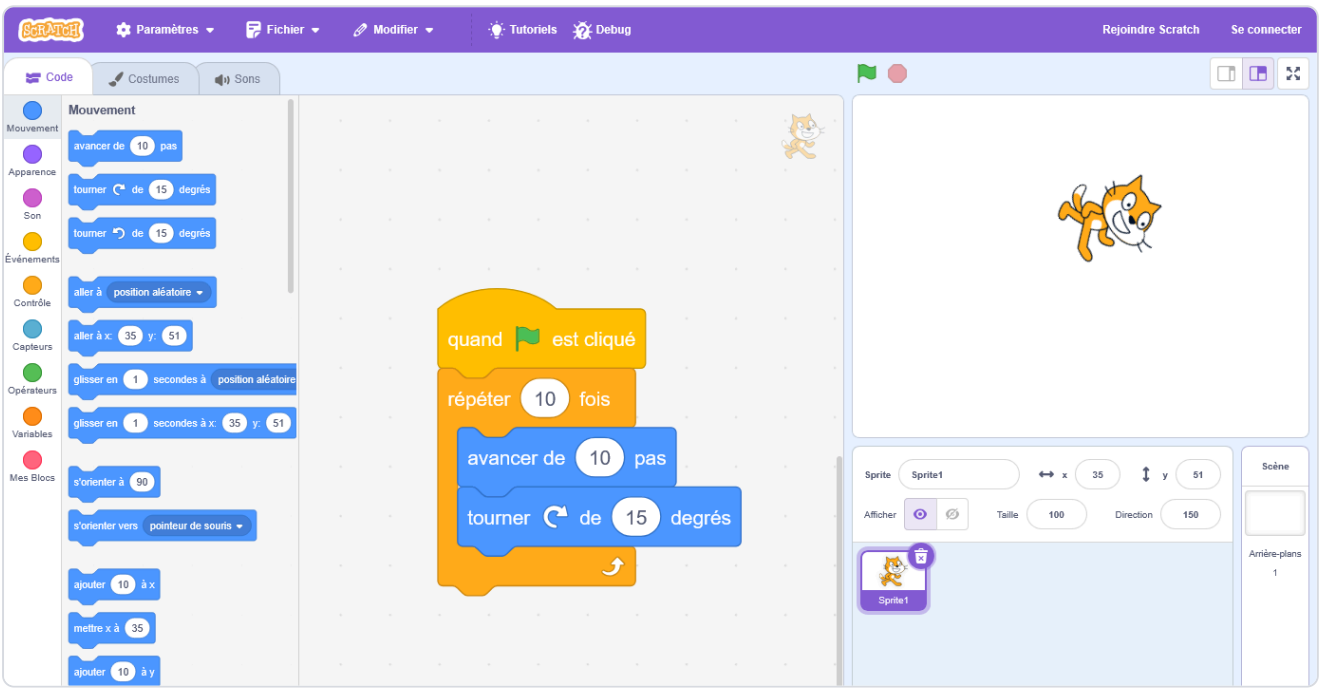
```
TANT QUE l'utilisateur n'appuie pas sur STOP
    jouer la chanson suivante
```

4. Programmation visuelle vs textuelle

Il existe deux façons d'écrire des programmes :

Programmation visuelle (Scratch)

On assemble des **blocs colorés** comme des Lego. Impossible de faire des erreurs de syntaxe !



Avantages : Facile, visuel, pas d'erreurs de frappe **Utilisé pour :** Apprendre la logique, Lego Mindstorm

Programmation textuelle (Python, C++...)

On écrit du **code** avec des mots et des symboles.

```python
for i in range(10):
    avancer(10)
    tourner(15)
```

Avantages : Plus puissant, utilisé professionnellement **Utilisé pour :** Arduino, drones Tello, applications réelles

La bonne nouvelle 🎉

Les concepts sont IDENTIQUES ! Une fois que tu maîtrises les variables, conditions et boucles en Scratch, tu peux les appliquer dans n'importe quel langage. Seule l'écriture change, pas la logique.

5. Notre parcours dans ce cours

```
   Scratch        →   Processing    →    Arduino - C++   →    Tello - Python
(blocs visuels)      (code simple)      (électronique)           (drones)
```

On commence par le visuel pour comprendre la **logique**, puis on passe progressivement au textuel pour faire des choses de plus en plus **puissantes** !

📝 Récapitulatif

Concept	C'est quoi ?	Analogie
Instruction	Un ordre simple	Une étape de recette
Variable	Une boîte qui stocke une valeur	Un tiroir étiqueté
Condition	Un choix selon une situation	Un embranchement sur la route
Boucle	Une répétition d'actions	Un refrain de chanson

Ces 4 concepts sont les **briques de base** de TOUT programme informatique. Maîtrise-les et tu pourras programmer n'importe quoi ! 🚀

Prochain chapitre : Scratch — Apprendre à coder en s'amusant

1.5 Introduction à Scratch

> 🎯 ***Objectif*** *: Découvrir Scratch, comprendre son histoire et sa philosophie, puis apprendre à l'utiliser via une autoformation en vidéo.*

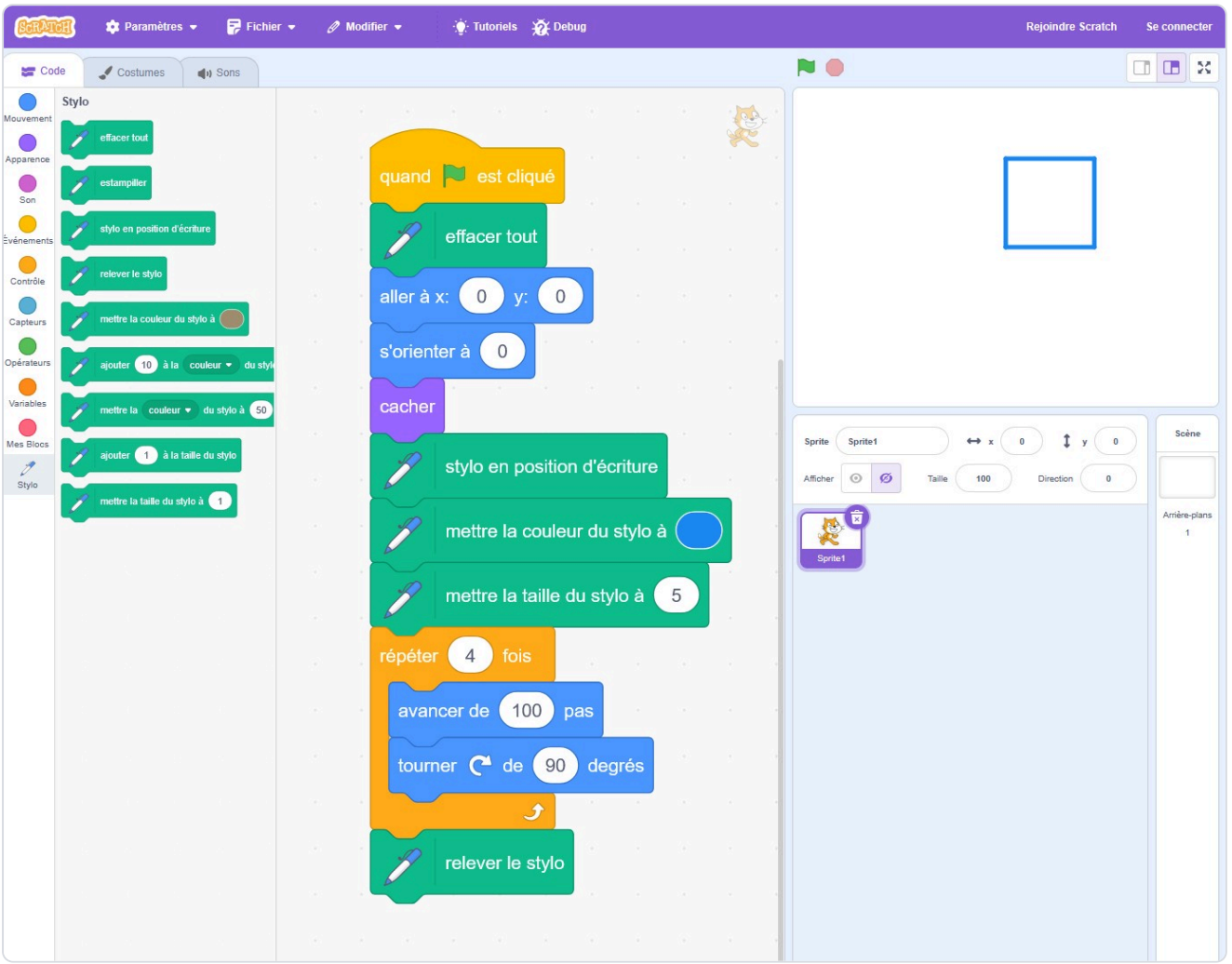
Slides du cours

- introduction scratch.pdf

1. C'est quoi Scratch ?

Un langage de programmation visuel

Scratch est un langage de programmation **gratuit** conçu spécialement pour apprendre à coder. Au lieu de taper du texte, on assemble des **blocs colorés** comme des pièces de puzzle.



Avantages :

- ✅ **Impossible de faire des erreurs de syntaxe** — les blocs s'emboîtent uniquement s'ils sont compatibles
- ✅ **Résultats immédiats** — tu vois tout de suite ce que ton programme fait
- ✅ **Créatif** — tu peux créer des jeux, des animations, des histoires interactives
- ✅ **Gratuit et en ligne** — rien à installer, ça marche dans le navigateur

🔗 **Accès** : https://scratch.mit.edu

2. L'histoire de Scratch

Qui l'a créé ?

Scratch a été créé par **Mitchel Resnick** et son équipe du groupe **Lifelong Kindergarten** (« Maternelle à vie ») au **MIT Media Lab** (Massachusetts Institute of Technology, États-Unis).

Info	Détail
🏛️ Créateur	Mitchel Resnick & Lifelong Kindergarten Group
🎓 Institution	MIT Media Lab (Boston, USA)
📅 Lancement	Mai 2007
🔢 Version actuelle	Scratch 3.0 (depuis 2019)
🌍 Utilisateurs	+100 millions dans le monde
💰 Prix	Gratuit, open source

Pourquoi Scratch a été créé ?

Mitchel Resnick était frustré : les enfants utilisaient les ordinateurs pour **consommer** du contenu (jeux, vidéos, réseaux sociaux) mais rarement pour **créer**.

Il voulait un outil qui permette à tout le monde — même sans connaissances techniques — de devenir **créateur** plutôt que simple consommateur.

> *« Nous voulons donner à chacun (de tous âges et de tous horizons) le pouvoir de créer ses propres histoires interactives, jeux et animations — et de les partager avec les autres. » — Mitchel Resnick*

La philosophie : "Low Floor, Wide Walls, High Ceiling"

L'équipe du MIT a conçu Scratch avec trois principes :

Principe	Signification
🚪 **Low Floor** (Seuil bas)	Facile de commencer, même pour un débutant total
🧱 **Wide Walls** (Murs larges)	Plein de possibilités : jeux, art, musique, histoires...
🏗️ **High Ceiling** (Plafond haut)	On peut faire des projets très sophistiqués

Les ancêtres de Scratch

Scratch n'est pas né de nulle part ! Il s'inspire de :

- **Logo** (1967) — le premier langage éducatif, avec sa célèbre « tortue » qui dessine
- **Squeak/Etoys** — développé par Alan Kay
- **Lego Mindstorms** — d'ailleurs, Mitchel Resnick a aussi travaillé dessus !

3. Ce que Scratch t'apprend (au-delà du code !)

Dans son **TED Talk** (lien en fin de chapitre), Mitchel Resnick explique que programmer avec Scratch développe des compétences qui dépassent largement l'informatique. Ces compétences sont utiles **dans tous les domaines de la vie** !

Les compétences développées

Compétence	Comment Scratch la développe
🎨 **Créativité**	Tu inventes des projets, tu exprimes tes idées de façon unique
🧩 **Pensée systémique**	Tu décomposes un problème complexe en petites étapes
🤝 **Collaboration**	Tu partages tes projets, tu remixes ceux des autres
💬 **Communication**	Tu apprends à exprimer tes idées clairement (pour que l'ordinateur comprenne !)
🔄 **Itération**	Tu testes, tu corriges, tu améliores — tu apprends de tes erreurs
🎯 **Persévérance**	Quand ça marche pas, tu cherches pourquoi et tu réessaies

L'analogie de Mitchel Resnick

> *« Apprendre à coder, ce n'est pas juste pour devenir programmeur. C'est comme apprendre à écrire : on n'apprend pas à écrire pour devenir écrivain professionnel, mais parce que c'est utile dans tous les aspects de la vie. »*

Quand tu programmes, tu ne fais pas que donner des ordres à un ordinateur. Tu apprends à **structurer ta pensée**, à **résoudre des problèmes**, et à **exprimer des idées complexes** de façon claire.

Ces compétences te serviront que tu deviennes ingénieur, artiste, médecin, entrepreneur, ou n'importe quoi d'autre !

4. Pourquoi apprendre Scratch dans ce cours ?

Le lien avec les robots et les drones 🤖🚁

Tu te demandes peut-être : « *Pourquoi apprendre Scratch si on va programmer des robots et des drones ?* »

Excellente question ! Voici pourquoi :

Concept Scratch	Application Robot/Drone
Coordonnées X, Y	Position du robot sur une grille, trajectoire du drone
Angles et rotations	Orientation du robot, virage du drone
Boucles (répéter)	Faire avancer un robot de 10 pas
Conditions (si/alors)	Si obstacle détecté, alors tourner
Variables	Stocker la distance parcourue, le niveau de batterie

En Scratch, tu vas apprendre :

- Les **coordonnées cartésiennes** (X, Y) — exactement ce qu'utilisent les robots et drones pour se repérer
- Les **angles** (0°, 90°, 180°...) — essentiels pour faire tourner un robot ou un drone
- La **géométrie** — tracer des formes, calculer des distances
- La **logique algorithmique** — les mêmes boucles et conditions qu'en Python ou C++

> 💡 ***En résumé*** *: Scratch te permet d'apprendre la* ***logique*** *de la programmation de façon visuelle et fun, avant de passer au code textuel avec Arduino et Python.*

5. Comment apprendre Scratch

Tu as le choix entre **deux parcours** selon tes préférences. Les deux mènent au même résultat !

Option A : Parcours Vidéo (recommandé pour débuter) 🎬

Autoformation du RÉCIT Québec — « Premiers pas avec Scratch pour tous »

Info	Détail
🏛️ Organisme	**RÉCIT MST** (Réseau pour le développement des Compétences des élèves par l'Intégration des Technologies)
🌍 Pays	Québec, Canada

Info	Détail
📚 Type	Service du Ministère de l'Éducation du Québec
⏱️ Durée	2 à 3 heures
💰 Prix	Gratuit
🔗 Lien	https://campus.recit.qc.ca/course/view.php?id=177

Structure du cours :

1. **Démarrer avec Scratch 3.0**
   - Création d'un compte
   - Interface et fonctionnalités de base
   - Premier défi accompagné (avec vidéo guidée)
2. **Donner vie à une animation**
   - Interactions entre personnages (dialogues)
   - Création de questions avec variables
3. **Créer du contenu**
   - Contenus animés (univers social, histoire...)
   - Questionnaires interactifs
   - Contenus artistiques avec le stylo
4. **Défis finaux** (3 défis à réaliser)
   - 🎆 **Feu d'artifice** — Tracer 250 segments colorés depuis le centre vers des points aléatoires
   - 🌀 **Spirale** — Dessiner une spirale de 30 segments qui grandissent (10, 20, 30 pixels...)
   - ❄️ **Flocons de neige** — Faire tomber des flocons du ciel avec des clones

> ⚠️ ***Note importante*** *: Ce cours a été conçu pour les enseignants québécois. Les* ***questionnaires et badges ne sont pas accessibles*** *sans compte canadien, mais* ***toutes les vidéos et tous les défis sont en accès libre*** *— c'est tout ce dont tu as besoin !*

Option B : Parcours Écrit (pour ceux qui préfèrent lire) 📖

Livre « Scratch au Collège » par Exo7

Info	Détail
📝 Auteur principal	**Arnaud Bodin** (Université de Lille)
🏛️ Projet	**Exo7** — Projet collaboratif français de ressources mathématiques

Info	Détail
🌍 Soutien	Université Lille 1, Unisciel, SMAI, SMF
📚 Type	PDF gratuit, licence Creative Commons
🔗 Téléchargement	https://exo7math.github.io/scratch3-exo7/ http://exo7.emath.fr/cours/livre-scratch.pdf
🌐 Site du projet	http://exo7.emath.fr

À propos d'Exo7 : Exo7 est un projet universitaire français qui propose des cours et exercices de mathématiques gratuits. Lancé par des enseignants de l'Université de Lille, il est soutenu par Unisciel (Université des Sciences en Ligne) et hébergé par la SMAI (Société de Mathématiques Appliquées et Industrielles) et la SMF (Société Mathématique de France). C'est une référence dans le monde francophone !

Structure du livre :

Partie I — Activités Scratch

1. Premiers pas
2. Répéter (boucles)
3. Coordonnées x, y
4. Si ... alors ... (conditions)
5. Entrée/Sortie
6. Variables et hasard
7. Si ... alors ... sinon ...
8. Plusieurs lutins
9. Sons
0. Et plus encore...

Partie II — Fiches (théorie)

Partie III — Activités sur feuille (algorithmes sans ordi)

> 💡 *Ce parcours est **moins progressif** que les vidéos mais couvre plus de contenu. Idéal si tu apprends vite ou si tu préfères lire plutôt que regarder des vidéos.*

6. Ce qu'on attend de toi

Objectif minimum ✅

1. **Suivre l'autoformation vidéo** (ou lire le PDF Exo7)
2. **Compléter au moins UN des 3 défis finaux** du cours RÉCIT :

- 🎆 Défi 1 : Feu d'artifice
- 🌀 Défi 2 : Spirale
- ❄️ Défi 3 : Flocons de neige

Comment ça se passe en cours

- 🎧 Tu travailles **à ton rythme** avec les vidéos ou le PDF
- 🙋 Le prof est là pour **t'aider** si tu bloques ou si tu ne comprends pas quelque chose
- 🤝 Tu peux **collaborer** avec tes voisins
- 💾 Tu **sauvegardes** tes projets sur ton compte Scratch

Conseils pour réussir

1. **Crée ton compte Scratch** dès le début → tu pourras sauvegarder et partager tes projets
2. **Ne saute pas les étapes** → les défis finaux utilisent tout ce que tu apprends avant
3. **Expérimente !** → Scratch est fait pour ça, tu ne peux rien casser
4. **Utilise la recherche Scratch** → Si tu bloques, cherche des projets similaires pour t'inspirer

7. Après Scratch : la suite du parcours

Une fois Scratch maîtrisé, tu auras acquis les bases pour :

```
Scratch (blocs)
    ↓
Exercices de géométrie et maths ← Chapitre suivant !
    ↓
Processing (premier code textuel)
    ↓
Arduino C++ (électronique)
    ↓
Python + Tello (drones)
```

Les **exercices de géométrie** qu'on fera dans le chapitre suivant te prépareront directement à la programmation de robots et drones. Tu verras, les coordonnées X/Y et les angles seront tes meilleurs amis ! 📐

📚 Ressources

Liens essentiels

Ressource	Lien
🐱 **Scratch** (site officiel)	https://scratch.mit.edu
🎬 **Autoformation RÉCIT** (vidéos)	https://campus.recit.qc.ca/course/view.php?id=177
📖 **Livre Exo7** (PDF à télécharger)	http://exo7.emath.fr/cours/livre-scratch.pdf

Pour aller plus loin

Ressource	Lien
🎥 **TED Talk de Mitchel Resnick**	https://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code
📖 Livre « Lifelong Kindergarten »	Par Mitchel Resnick (MIT Press, en anglais)
🌐 Scratch Wiki	https://en.scratch-wiki.info
🏛️ Site Exo7 (autres ressources maths)	http://exo7.emath.fr

> 💡 ***Le TED Talk est vraiment à voir !*** *En 15 minutes, Mitchel Resnick explique pourquoi apprendre à coder est important pour tout le monde, pas seulement pour les futurs informaticiens. Il donne des exemples concrets de projets créés par des jeunes et montre comment la programmation développe la créativité et la pensée logique.*

Prochain chapitre : Exercices Scratch — Mathématiques et Géométrie pour la robotique

1.6 Exercices Scratch — Série 1 : Variables et Calculs

> 🎯 ***Objectif*** *: Apprendre à utiliser les* ***variables*** *dans Scratch pour stocker des valeurs et faire des calculs.*

🎬 Exercice 1 : Somme de 2 nombres (guidé)

Consigne

Regarde la vidéo et reproduis le programme qui :

1. Demande à l'utilisateur d'entrer un premier nombre

2. Demande un deuxième nombre
3. Calcule la somme des deux
4. Affiche le résultat

Ressource

📺 **Vidéo tutoriel** : https://www.youtube.com/watch?v=Zn8VLULg7zQ

Résultat attendu

Le chat Scratch doit :

- Dire "Bonjour! Voici un programme qui calcule la somme de 2 nombres"
- Demander "Quel est le nombre 1?"
- Demander "Quel est le nombre 2?"
- Afficher "Voici la somme: X" (où X est le résultat)

Ce que tu vas apprendre

- Créer des **variables** (nombre 1, nombre 2, somme)
- Utiliser le bloc **"demander ... et attendre"**
- Récupérer la **réponse** de l'utilisateur
- Faire une **addition** avec les opérateurs
- **Regrouper** du texte et des variables pour l'affichage

✏️ Exercice 2 : Produit de 2 nombres

Consigne

Adapte le programme de l'exercice 1 pour calculer le **produit** (multiplication) de 2 nombres au lieu de la somme.

Résultat attendu

```yaml
Chat : "Bonjour! Voici un programme qui calcule le produit de 2 nombres"
Chat : "Quel est le nombre 1?" → Utilisateur entre: 7
Chat : "Quel est le nombre 2?" → Utilisateur entre: 8
Chat : "Voici le produit: 56"
```

Pseudo-code

```
QUAND drapeau vert cliqué

DIRE "Bonjour! Voici un programme qui calcule le produit de 2 nombres" PENDANT 3
secondes

DEMANDER "Quel est le nombre 1?" ET ATTENDRE
METTRE [nombre 1] À (réponse)

DEMANDER "Quel est le nombre 2?" ET ATTENDRE
METTRE [nombre 2] À (réponse)

METTRE [produit] À (nombre 1 * nombre 2)

DIRE (REGROUPER "Voici le produit: " ET (produit)) PENDANT 4 secondes
```

💡 Indice

La seule différence avec l'exercice 1, c'est :

- Le texte affiché (remplacer "somme" par "produit")
- L'opération : utiliser **×** (multiplication) au lieu de **+** (addition)
- Le nom de la variable résultat : "produit" au lieu de "somme"

✏️ Exercice 3 : Somme de 3 nombres

Consigne

Adapte le programme pour calculer la somme de **3 nombres** au lieu de 2.

Résultat attendu

YAML

```
Chat : "Bonjour! Voici un programme qui calcule la somme de 3 nombres"
Chat : "Quel est le nombre 1?" → Utilisateur entre: 10
Chat : "Quel est le nombre 2?" → Utilisateur entre: 20
Chat : "Quel est le nombre 3?" → Utilisateur entre: 30
Chat : "Voici la somme: 60"
```

Pseudo-code

```
QUAND drapeau vert cliqué

DIRE "Bonjour! Voici un programme qui calcule la somme de 3 nombres" PENDANT 3
secondes

DEMANDER "Quel est le nombre 1?" ET ATTENDRE
METTRE [nombre 1] À (réponse)

DEMANDER "Quel est le nombre 2?" ET ATTENDRE
METTRE [nombre 2] À (réponse)

DEMANDER "Quel est le nombre 3?" ET ATTENDRE
METTRE [nombre 3] À (réponse)

METTRE [somme] À (nombre 1 + nombre 2 + nombre 3)

DIRE (REGROUPER "Voici la somme: " ET (somme)) PENDANT 4 secondes
```

💡 Indices

1. Tu dois créer une **3ème variable** : "nombre 3"
2. Tu dois ajouter un **3ème bloc "demander"** pour récupérer le 3ème nombre
3. Pour additionner 3 nombres dans Scratch, tu dois **imbriquer** les opérateurs :

   ```
   ( (nombre 1 + nombre 2) + nombre 3 )
   ```

   C'est-à-dire : mettre un bloc "+" dans un autre bloc "+"

✏️ Exercice 4 : Moyenne de 4 nombres

Consigne

Crée un programme qui calcule la **moyenne** de 4 nombres.

> 📐 ***Rappel mathématique*** *: La moyenne de N nombres = (somme de tous les nombres) ÷ N*

Résultat attendu

```yaml
Chat : "Bonjour! Voici un programme qui calcule la moyenne de 4 nombres"
Chat : "Quel est le nombre 1?" → Utilisateur entre: 10
Chat : "Quel est le nombre 2?" → Utilisateur entre: 12
Chat : "Quel est le nombre 3?" → Utilisateur entre: 14
Chat : "Quel est le nombre 4?" → Utilisateur entre: 16
Chat : "Voici la moyenne: 13"
```

Pseudo-code

```
QUAND drapeau vert cliqué

DIRE "Bonjour! Voici un programme qui calcule la moyenne de 4 nombres" PENDANT 3
secondes

DEMANDER "Quel est le nombre 1?" ET ATTENDRE
METTRE [nombre 1] À (réponse)

DEMANDER "Quel est le nombre 2?" ET ATTENDRE
METTRE [nombre 2] À (réponse)

DEMANDER "Quel est le nombre 3?" ET ATTENDRE
METTRE [nombre 3] À (réponse)

DEMANDER "Quel est le nombre 4?" ET ATTENDRE
METTRE [nombre 4] À (réponse)

METTRE [somme] À (nombre 1 + nombre 2 + nombre 3 + nombre 4)
METTRE [moyenne] À (somme / 4)

DIRE (REGROUPER "Voici la moyenne: " ET (moyenne)) PENDANT 4 secondes
```

💡 Indices

1. Tu as besoin de **6 variables** : nombre 1, nombre 2, nombre 3, nombre 4, somme, moyenne
2. Calcule d'abord la **somme** des 4 nombres
3. Puis **divise** cette somme par 4 pour obtenir la moyenne
4. Pour additionner 4 nombres, tu dois imbriquer 3 blocs "+" :

```
( ( (nombre 1 + nombre 2) + nombre 3) + nombre 4 )
```

⚠️ Attention aux décimales !

Si tu entres 10, 11, 12, 13, la moyenne sera 11.5 (avec une décimale). C'est normal ! Scratch gère les nombres décimaux automatiquement.

🏆 Défi Bonus : Calculatrice complète

Consigne

Crée un programme qui :

1. Demande 2 nombres
2. Demande quelle opération faire (+, -, ×, ÷)
3. Affiche le résultat de l'opération choisie

Résultat attendu

```bash
Chat : "Bienvenue dans la calculatrice!"
Chat : "Quel est le premier nombre?" → 15
Chat : "Quel est le deuxième nombre?" → 3
Chat : "Quelle opération? (+, -, *, /)" → *
Chat : "Résultat: 15 * 3 = 45"
```

Pseudo-code

```
QUAND drapeau vert cliqué

DIRE "Bienvenue dans la calculatrice!" PENDANT 2 secondes

DEMANDER "Quel est le premier nombre?" ET ATTENDRE
METTRE [nombre 1] À (réponse)

DEMANDER "Quel est le deuxième nombre?" ET ATTENDRE
METTRE [nombre 2] À (réponse)

DEMANDER "Quelle opération? (+, -, *, /)" ET ATTENDRE
METTRE [opération] À (réponse)

SI (opération = "+") ALORS
    METTRE [résultat] À (nombre 1 + nombre 2)
FIN SI

SI (opération = "-") ALORS
    METTRE [résultat] À (nombre 1 - nombre 2)
FIN SI

SI (opération = "*") ALORS
    METTRE [résultat] À (nombre 1 * nombre 2)
FIN SI

SI (opération = "/") ALORS
    METTRE [résultat] À (nombre 1 / nombre 2)
FIN SI

DIRE (REGROUPER "Résultat: " ET (nombre 1) ET " " ET (opération) ET " " ET (nombre 2)
ET " = " ET (résultat)) PENDANT 5 secondes
```

💡 Indices

- Tu auras besoin du bloc **"si ... alors"** (catégorie Contrôle, orange)
- Tu auras besoin du bloc **"... = ..."** (catégorie Opérateurs, vert) pour comparer l'opération
- Le bloc "regrouper" peut s'imbriquer plusieurs fois pour créer un long message

📚 Ressources

Si tu es bloqué(e)

1. **Demande au prof !** C'est pour ça qu'il est là 🙋
2. **Relis le pseudo-code** ligne par ligne et traduis en blocs Scratch

3. **Cherche sur Scratch** des projets similaires pour t'inspirer (mais ne copie pas bêtement !)

Tutoriels Scratch (en dernier recours)

⚠️ **Attention** : Ces tutoriels contiennent parfois les solutions. Essaie d'abord par toi-même !

🔗 https://youtube.com/playlist?list=PLbE85KlaADWnn-vHutGey_JJEAwpkFsGE&si=1ZMTLjVzJ0VteRco

✅ Checklist de validation

Avant de passer à la suite, vérifie que tu as bien compris :

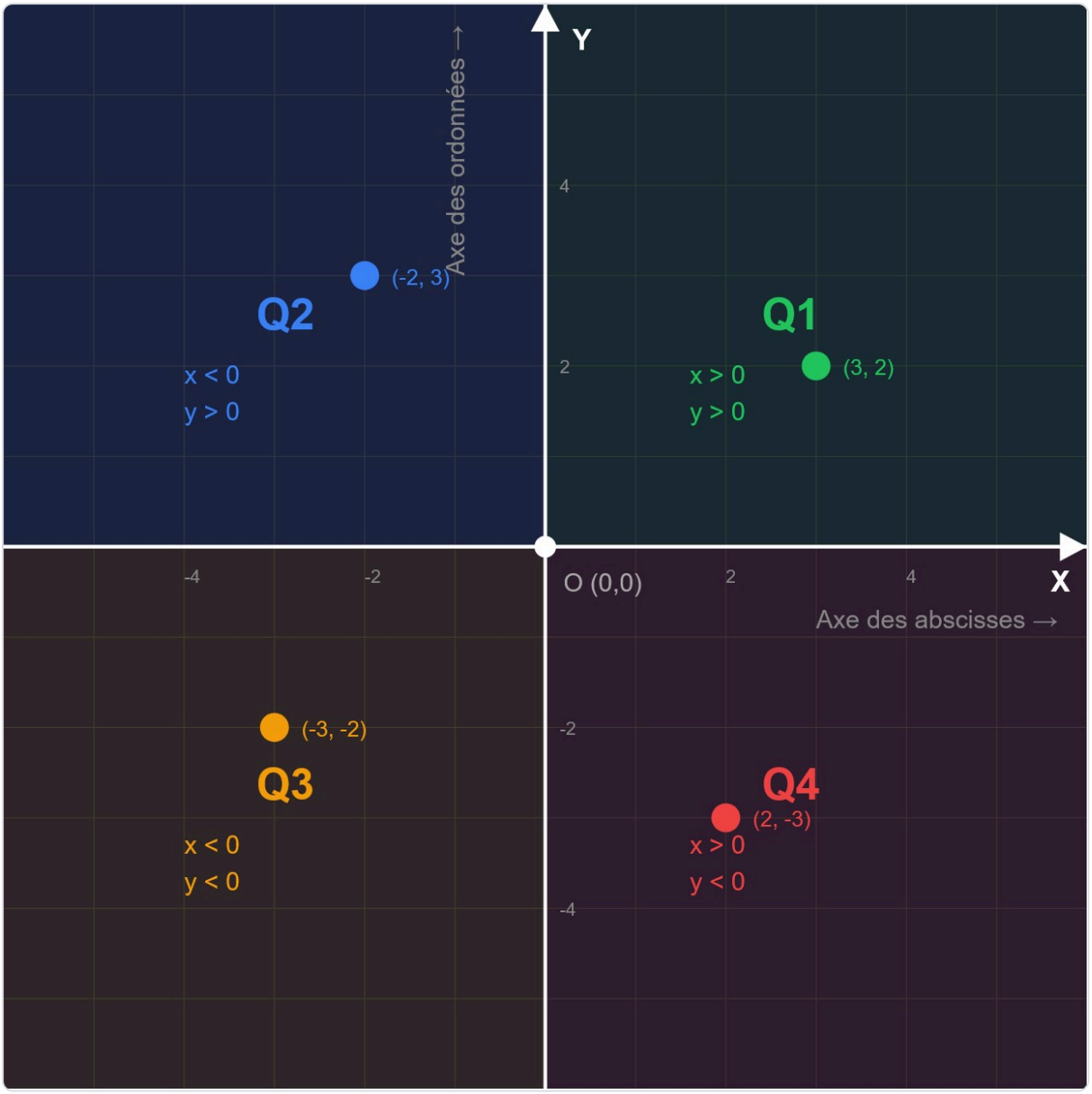
Concept	Je sais faire
Créer une variable	☐
Demander une valeur à l'utilisateur	☐
Stocker la réponse dans une variable	☐
Faire un calcul (+, -, ×, ÷)	☐
Afficher du texte avec une variable	☐
Imbriquer des opérateurs	☐

Série suivante : Le plan cartésien et les coordonnées (X, Y)

1.7 Exercices Scratch — Série 2 : Le Plan Cartésien

> 🎯 ***Objectif*** *: Comprendre le système de coordonnées (X, Y) utilisé par Scratch — et par TOUS les robots et drones !*

📐 Rappel : Le Plan Cartésien



Le plan cartésien est divisé en **4 quadrants** + des cas spéciaux :

Zone	Condition sur X	Condition sur Y	Exemple
Quadrant 1 (Q1)	x > 0 (positif)	y > 0 (positif)	(3, 2)
Quadrant 2 (Q2)	x < 0 (négatif)	y > 0 (positif)	(-2, 3)
Quadrant 3 (Q3)	x < 0 (négatif)	y < 0 (négatif)	(-3, -2)
Quadrant 4 (Q4)	x > 0 (positif)	y < 0 (négatif)	(2, -3)
Origine	x = 0	y = 0	(0, 0)
Axe des ordonnées	x = 0	y ≠ 0	(0, 5)

Zone	Condition sur X	Condition sur Y	Exemple
Axe des abscisses	x ≠ 0	y = 0	(7, 0)

🤖 Pourquoi c'est important pour les robots ?

Dans Scratch, le chat se déplace sur une scène qui EST un plan cartésien :

- **X** va de **-240** (gauche) à **+240** (droite)
- **Y** va de **-180** (bas) à **+180** (haut)
- Le **centre** de la scène est l'origine **(0, 0)**

C'est EXACTEMENT le même système qu'utilisent les drones et robots pour se repérer dans l'espace ! 🚁

✏️ Exercice : Détecteur de Quadrant

Consigne

Crée un programme qui :

1. Demande à l'utilisateur une coordonnée **X**
2. Demande une coordonnée **Y**
3. Analyse les valeurs et dit dans **quelle zone** du plan se trouve le point

Résultat attendu

```
Chat : "Donne-moi les coordonnées d'un point et je te dirai où il se situe."

Chat : "Quelle est la valeur de x?" → Utilisateur entre: 5
Chat : "Quelle est la valeur de y?" → Utilisateur entre: -3

Chat : "Le point (5, -3) est dans le quadrant 4"
```

Autres exemples de résultats :

- `(3, 7)` → "Le point (3, 7) est dans le quadrant 1"
- `(-2, 4)` → "Le point (-2, 4) est dans le quadrant 2"
- `(-5, -1)` → "Le point (-5, -1) est dans le quadrant 3"
- `(0, 0)` → "Le point (0, 0) est à l'origine du plan cartésien"
- `(0, 8)` → "Le point (0, 8) est sur l'axe des ordonnées"
- `(6, 0)` → "Le point (6, 0) est sur l'axe des abscisses"

Pseudo-code (Partie 1 : Introduction)

Cette partie est donnée complètement :

```
QUAND drapeau vert cliqué

METTRE [x] À 0
METTRE [y] À 0

DIRE "Donne-moi les coordonnées d'un point et je te dirai où il se situe." PENDANT 3
secondes

DEMANDER "Quelle est la valeur de x?" ET ATTENDRE
METTRE [x] À (réponse)

DEMANDER "Quelle est la valeur de y?" ET ATTENDRE
METTRE [y] À (réponse)

// ═══ ICI COMMENCENT LES CONDITIONS MYSTÉRIEUSES ═══ //
```

🧩 Pseudo-code (Partie 2 : Les Conditions Mystérieuses)

À toi de compléter les `???` ! Utilise le tableau des quadrants pour trouver les bonnes conditions.

```
// ═══════════════════════════════════════════════════
//  🟢 QUADRANT 1 — Le coin "tout positif"
// ═══════════════════════════════════════════════════

SI ( x ??? 0 ) ET ( y ??? 0 ) ALORS
    DIRE "Le point (...) est dans le quadrant 1" PENDANT 5 secondes
FIN SI


// ═══════════════════════════════════════════════════
//  🔵 QUADRANT 2 — X est parti se cacher à gauche...
// ═══════════════════════════════════════════════════

SI ( x ??? ) ET ( ??? ) ALORS
    DIRE "Le point (...) est dans le quadrant 2" PENDANT 5 secondes
FIN SI


// ═══════════════════════════════════════════════════
//  🟠 QUADRANT 3 — Le coin des déprimés (tout négatif)
// ═══════════════════════════════════════════════════

SI ( ??? ) ET ( ??? ) ALORS
    DIRE "Le point (...) est dans le quadrant 3" PENDANT 5 secondes
FIN SI


// ═══════════════════════════════════════════════════
//  🔴 QUADRANT 4 — X positif mais Y a le cafard
// ═══════════════════════════════════════════════════

SI ( ??? ) ET ( ??? ) ALORS
    DIRE "Le point (...) est dans le quadrant 4" PENDANT 5 secondes
FIN SI
```

🧩 Pseudo-code (Partie 3 : Les Cas Spéciaux)

Encore plus mystérieux ! Ces cas arrivent quand le point est **pile** sur un axe.

```
// ═══════════════════════════════════════════════
//  ⚪ L'ORIGINE — Le centre de tout !
//  Indice : les deux coordonnées sont... nulles
// ═══════════════════════════════════════════════

SI ( x ??? ) ET ( y ??? ) ALORS
    DIRE "Le point (...) est à l'origine du plan cartésien" PENDANT 5 secondes
FIN SI


// ═══════════════════════════════════════════════
//  📍 AXE DES ORDONNÉES — La ligne verticale
//  Indice : x vaut zéro, mais y peut être n'importe quoi (sauf zéro)
// ═══════════════════════════════════════════════

SI ( x ??? ) ET ( NON ( y ??? ) ) ALORS
    DIRE "Le point (...) est sur l'axe des ordonnées" PENDANT 5 secondes
FIN SI


// ═══════════════════════════════════════════════
//  📍 AXE DES ABSCISSES — La ligne horizontale
//  Indice : c'est l'inverse du cas précédent !
// ═══════════════════════════════════════════════

SI ( ??? ) ET ( ??? ) ALORS
    DIRE "Le point (...) est sur l'axe des abscisses" PENDANT 5 secondes
FIN SI
```

💡 Indices Progressifs

🔍 Indice 1 : Les symboles de comparaison

Dans Scratch, tu as accès aux blocs de comparaison (catégorie **Opérateurs**, vert clair) :

- `( ) > ( )` → plus grand que
- `( ) < ( )` → plus petit que
- `( ) = ( )` → égal à

🔍 Indice 2 : Combiner deux conditions

Pour vérifier deux choses en même temps, utilise le bloc **ET** :

```
( condition1 ) ET ( condition2 )
```

Ce bloc est dans la catégorie **Opérateurs** (vert clair).

🔍 Indice 3 : Le bloc "NON"

Le bloc **NON** inverse une condition :

- `NON (x = 0)` signifie "x est différent de 0"

C'est utile pour les axes ! (ex: sur l'axe Y, x=0 mais y≠0)

🔍 Indice 4 : Solution du Quadrant 1

```css
SI ( x > 0 ) ET ( y > 0 ) ALORS
```

Le quadrant 1, c'est quand X ET Y sont tous les deux **positifs** (supérieurs à zéro).

🔍 Indice 5 : Logique des quadrants

Quadrant	X	Y
Q1	> 0	> 0
Q2	< 0	> 0
Q3	< 0	< 0
Q4	> 0	< 0

Tu vois le pattern ? 🧠

⚠️ Attention à l'ordre !

L'ordre des conditions est **important** ! Si tu mets la condition de l'origine à la fin, elle ne sera peut-être jamais vérifiée...

Conseil : Teste ton programme avec ces valeurs pour vérifier que tout marche :

- `(5, 3)` → Q1
- `(-2, 4)` → Q2
- `(-3, -7)` → Q3
- `(8, -1)` → Q4
- `(0, 0)` → Origine
- `(0, 99)` → Axe Y
- `(42, 0)` → Axe X

🏆 Défi Bonus : Afficher le point sur la scène !

Une fois que ton programme détecte le quadrant, fais **bouger le chat** vers ce point !

Pseudo-code du bonus

```
// Après avoir affiché le quadrant...

ALLER À x: (x) y: (y)
DIRE "Me voilà!" PENDANT 2 secondes
```

💡 Indice

Le bloc **"aller à x: ... y: ..."** est dans la catégorie **Mouvement** (bleu).

⚠️ Attention : si l'utilisateur entre des valeurs trop grandes (ex: x=500), le chat sortira de l'écran ! La scène Scratch va de -240 à +240 en X, et de -180 à +180 en Y.

🏆 Défi Expert : Boucle infinie

Modifie ton programme pour qu'il demande des coordonnées **en boucle** jusqu'à ce que l'utilisateur dise "stop".

Pseudo-code

```
RÉPÉTER JUSQU'À ( continuer = "non" )

    // ... tout le code des questions et conditions ...

    DEMANDER "Veux-tu continuer? (oui ou non)" ET ATTENDRE
    METTRE [continuer] À (réponse)

FIN RÉPÉTER

DIRE "À bientôt!" PENDANT 2 secondes
```

✅ Checklist de validation

Concept	Je sais faire
Utiliser les comparaisons (>, <, =)	☐

Concept	Je sais faire
Combiner des conditions avec ET	☐
Utiliser NON pour inverser une condition	☐
Enchaîner plusieurs blocs SI	☐
Faire bouger un sprite aux coordonnées (x, y)	☐

🤖 Le lien avec les robots

Ce que tu viens de programmer, c'est exactement ce que fait un robot ou un drone pour se repérer !

Scratch	Robot réel
Coordonnée X	Position gauche/droite
Coordonnée Y	Position avant/arrière
`aller à x: y:`	Commande de déplacement
Quadrants	Zones de la pièce/du terrain

Quand tu programmeras le drone Tello en Python, tu utiliseras les mêmes concepts :

```python
drone.go_xyz_speed(x, y, z, speed)
```

Série suivante : Dessiner des figures géométriques avec le stylo

1.8 Exercices Scratch : Formes Géométriques 📐

> 🎯 ***Objectif*** *: Utiliser les blocs de dessin (stylo) de Scratch pour tracer des formes géométriques. Tu vas réviser tes maths tout en programmant !*

🧠 Rappel : Les blocs essentiels

Avant de commencer, assure-toi de connaître ces blocs de la catégorie **Stylo** (extension à ajouter) :

Bloc	Ce qu'il fait
`stylo en position d'écriture`	Le sprite commence à dessiner

Bloc	Ce qu'il fait
`relever le stylo`	Le sprite arrête de dessiner
`effacer tout`	Efface tous les dessins
`mettre la couleur du stylo à [...]`	Change la couleur
`mettre la taille du stylo à [...]`	Change l'épaisseur du trait
`ajouter [...] à la couleur du stylo`	Fait varier la couleur (arc-en-ciel!)

Et ces blocs de **Mouvement** :

Bloc	Ce qu'il fait
`avancer de [...] pas`	Avance tout droit
`tourner ↻ de [...] degrés`	Tourne à droite
`tourner ↺ de [...] degrés`	Tourne à gauche
`aller à x: [...] y: [...]`	Se positionne

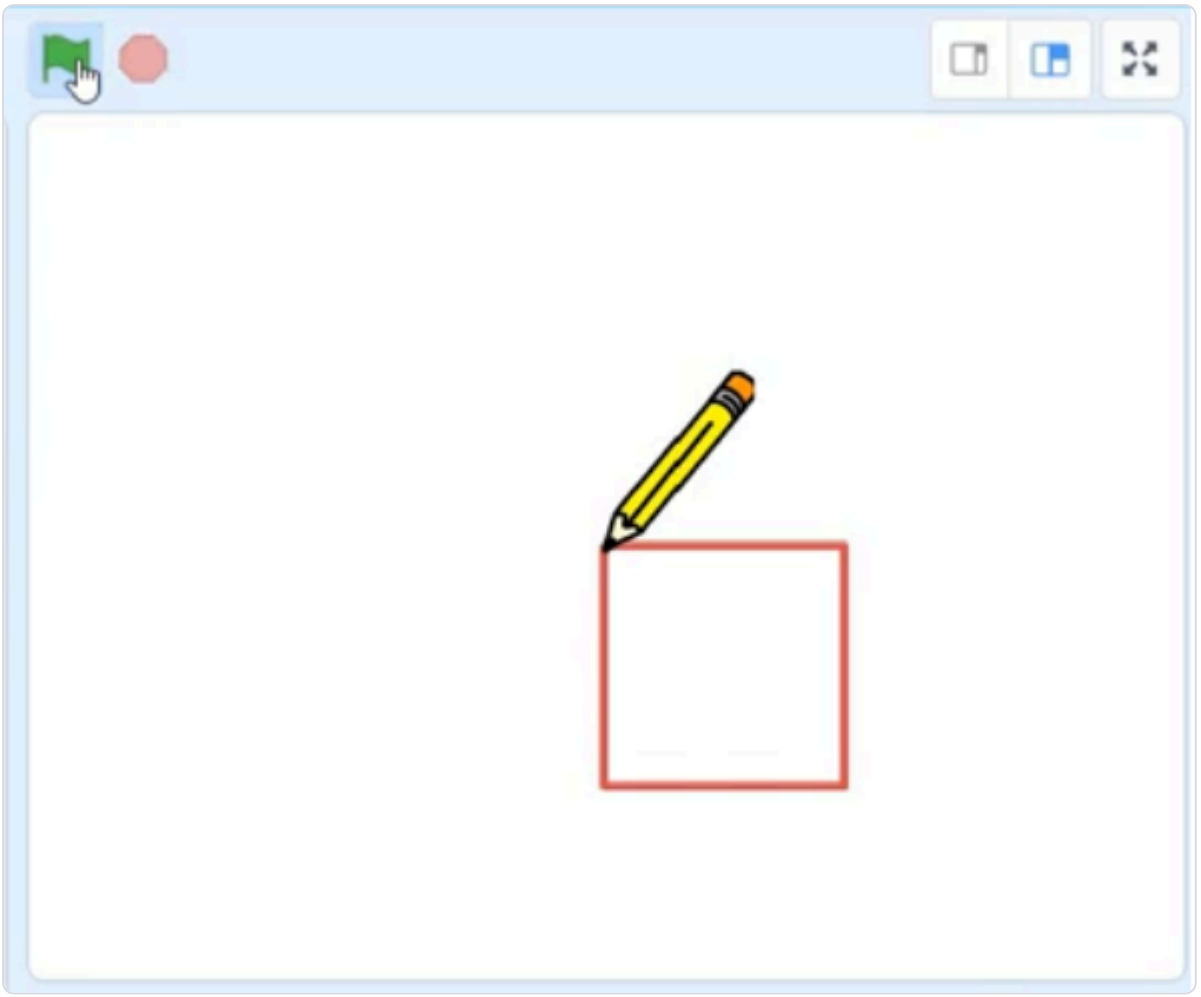
🟥 Exercice 1 : Le Carré

Difficulté : ⭐ Facile

Objectif

Dessiner un carré avec le stylo.

Ce que tu dois obtenir



Indice mathématique

- Un carré a **4 côtés égaux**
- Chaque angle est de **90°**
- Donc tu tournes de combien à chaque coin ? 🤔

Consigne

1. **Essaie d'abord par toi-même !** Réfléchis à la logique.
2. Si tu bloques, regarde la vidéo d'aide ci-dessous.

🎬 Vidéo d'aide

https://www.youtube.com/watch?v=yoEj7ah5Syc

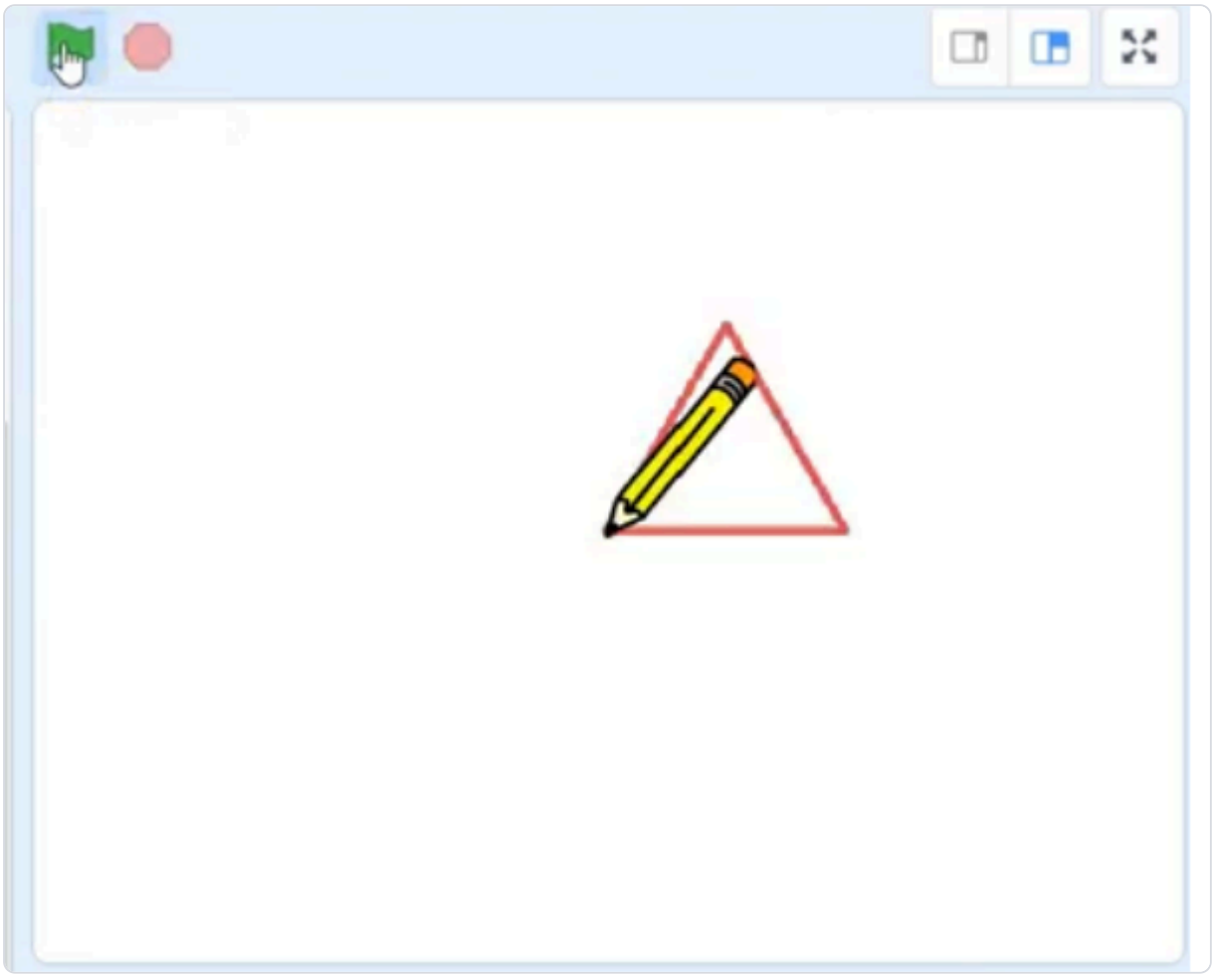
🔺 Exercice 2 : Le Triangle Équilatéral

Difficulté : ⭐ Facile

Objectif

Dessiner un triangle équilatéral (3 côtés égaux).

Ce que tu dois obtenir



Indice mathématique

- Un triangle équilatéral a **3 côtés égaux**
- La somme des angles intérieurs d'un triangle = **180°**
- Donc chaque angle intérieur = **60°**
- ⚠️ Mais attention ! L'angle de rotation n'est PAS l'angle intérieur... C'est l'**angle extérieur** !
- Angle extérieur = 180° - angle intérieur = ?

Consigne

1. **Essaie d'abord par toi-même !** C'est le même principe que le carré, mais avec 3 côtés.
2. Si tu bloques, regarde la vidéo d'aide.

🎬 Vidéo d'aide

https://www.youtube.com/watch?v=Ju9WvQgktzM

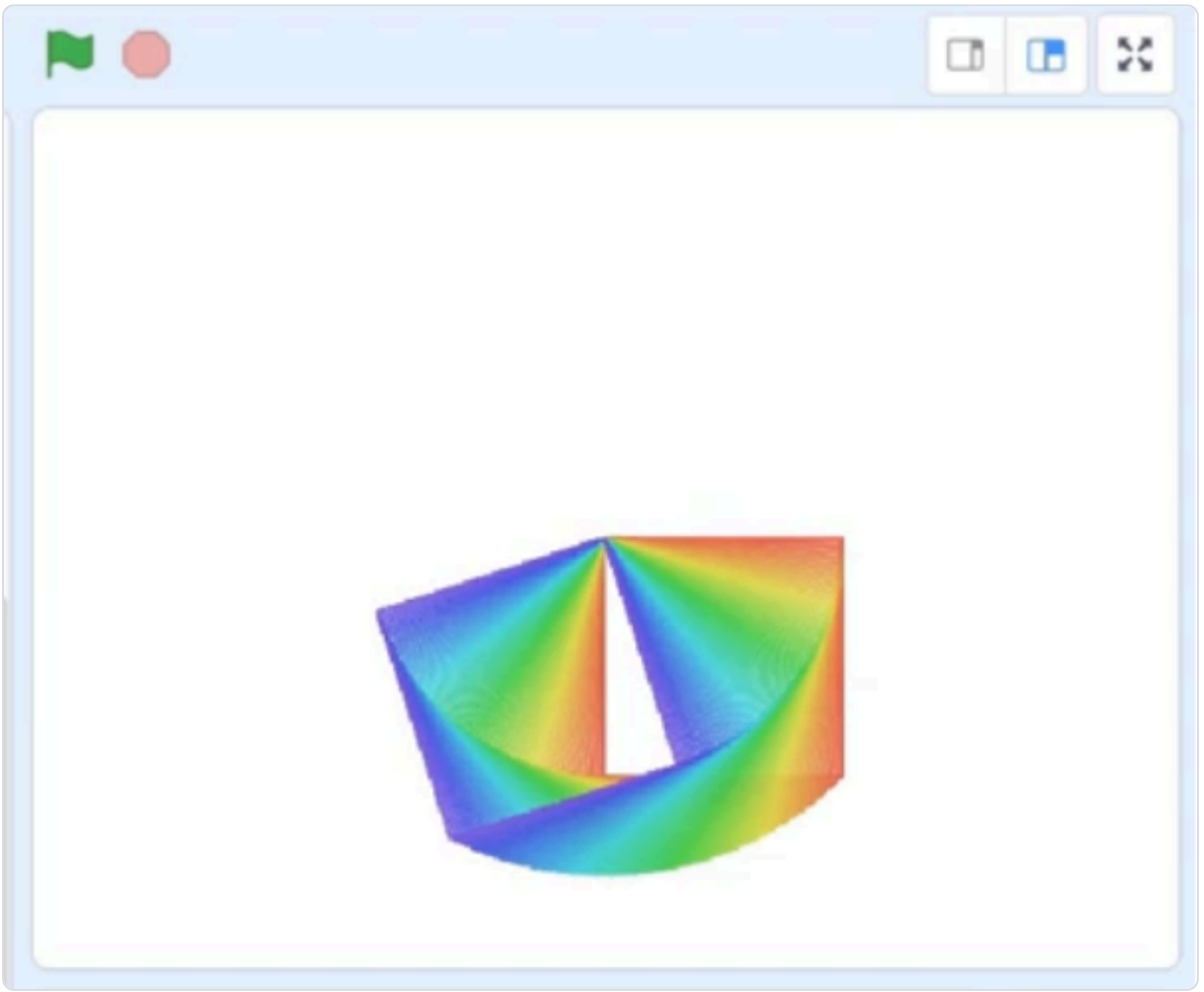
🌈 Exercice 3 : Le Carré Arc-en-ciel

Difficulté : ⭐⭐ Moyen

Objectif

Dessiner plusieurs carrés qui tournent avec des couleurs qui changent pour créer un effet arc-en-ciel.

Ce que tu dois obtenir



Indices

- Tu vas avoir besoin d'une **boucle dans une boucle** (boucle imbriquée)
- Boucle extérieure : répéter le carré plusieurs fois en tournant un peu à chaque fois
- Boucle intérieure : dessiner un carré (comme l'exercice 1)
- Utilise `ajouter [...] à la couleur du stylo` pour faire varier les couleurs

Consigne

1. **Essaie d'abord par toi-même !** Pars de ton code du carré et ajoute une boucle autour.
2. Si tu bloques, regarde la vidéo d'aide.

🎬 Vidéo d'aide

https://www.youtube.com/watch?v=ZxY2I8sWyRY

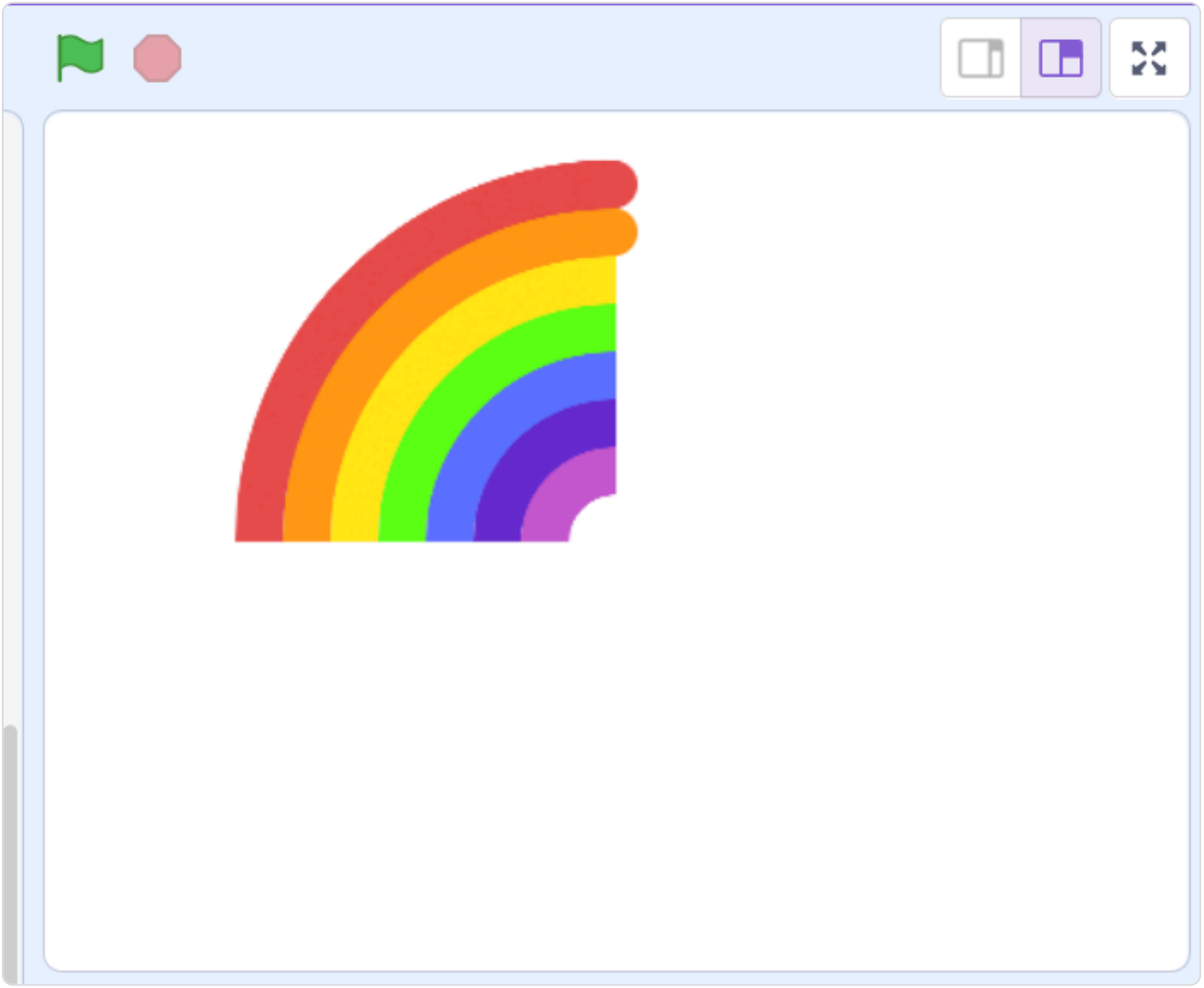
🌈 Exercice 4 : L'Arc-en-ciel (Emoji Style)

Difficulté : ⭐⭐ Moyen

Objectif

Dessiner un arc-en-ciel comme l'emoji 🌈 avec des demi-cercles concentriques de couleurs différentes.

Ce que tu dois obtenir



Indices

- Un demi-cercle = tourner de **180°** au total
- Mais tu le fais en plein de petits pas (ex: répéter 180 fois : avancer de 2, tourner de 1°)
- Pour faire plusieurs arcs concentriques, tu changes la **taille du stylo** (épaisseur)
- Les couleurs de l'arc-en-ciel : rouge, orange, jaune, vert, bleu, indigo, violet

Consigne

1. **Essaie d'abord par toi-même !** Commence par tracer UN demi-cercle.
2. Si tu bloques, regarde la vidéo d'aide.

🎬 Vidéo d'aide

https://www.youtube.com/watch?v=2eOPLA_z-Ak

🏆 Défi : Construction de Quadrilatères

Difficulté : ⭐⭐⭐ Avancé

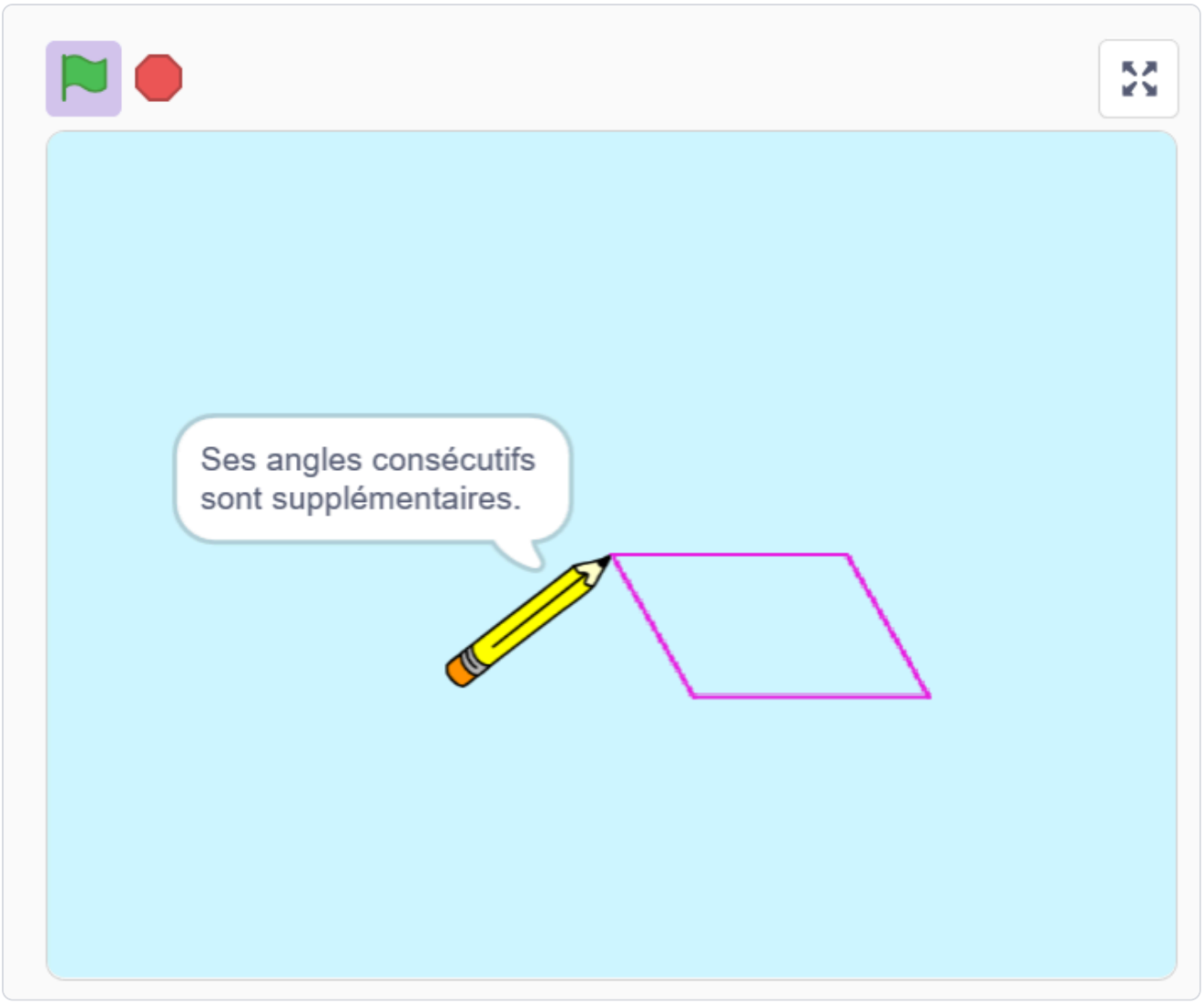
Objectif

Construire différents quadrilatères à partir de leurs propriétés mathématiques.

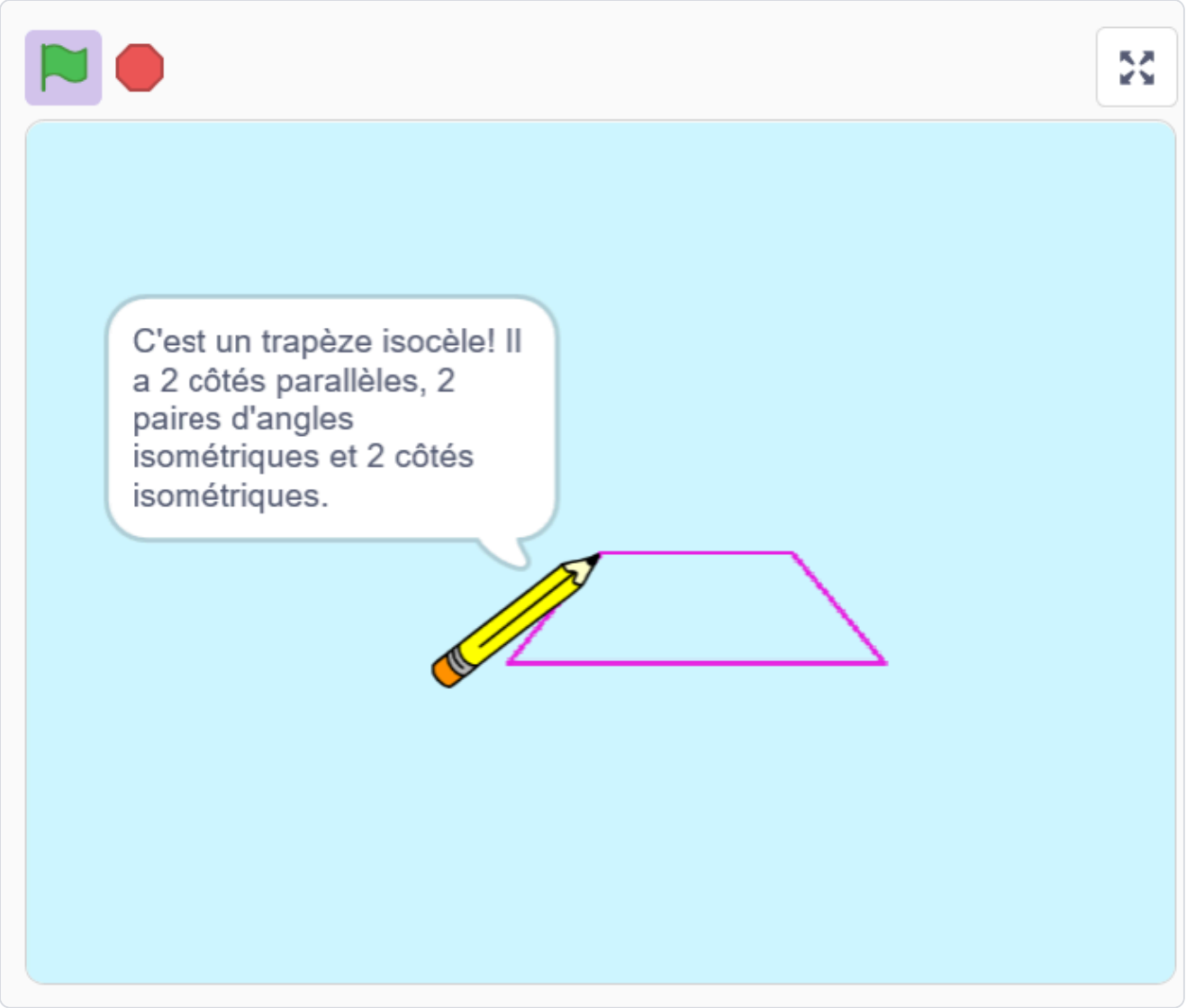
Menu des formes disponibles



Parallèlogramme



Trapèze



Point de départ

👉 **Projet à remixer** : scratch.mit.edu/projects/218975967

Les quadrilatères à maîtriser

Forme	Propriétés
Carré	4 côtés égaux, 4 angles de 90°
Rectangle	Côtés opposés égaux, 4 angles de 90°
Losange	4 côtés égaux, angles opposés égaux
Parallélogramme	Côtés opposés parallèles et égaux
Trapèze isocèle	2 côtés parallèles, 2 côtés non-parallèles égaux

Forme	Propriétés
Trapèze rectangle	2 côtés parallèles, 2 angles droits

Consigne

Explore le projet de départ et essaie de comprendre comment chaque forme est construite à partir de ses propriétés.

⭐ Défi : Une Étoile Mathématique

Difficulté : ⭐⭐⭐ Avancé

Objectif

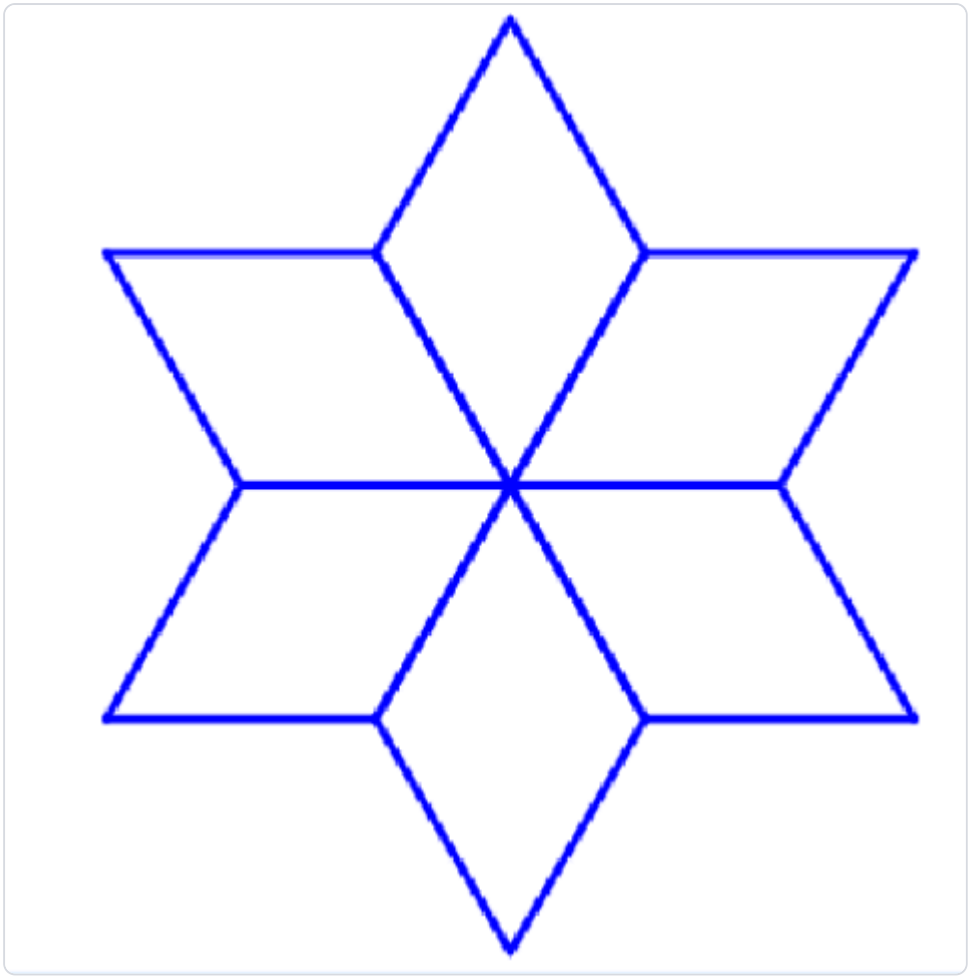
Dessiner des étoiles en utilisant les angles et les propriétés géométriques.

Ce que tu dois obtenir

Étoile à 5 branches (pentagramme)



Étoile à 6 branches (hexagramme avec losanges)



Indice mathématique

Pour l'étoile à 5 branches :

- Tu traces 5 segments qui se croisent
- L'angle de rotation = **144°**
- C'est une seule boucle de 5 répétitions !

Pour l'étoile à 6 branches (avec losanges) :

- C'est 6 losanges disposés en cercle
- Chaque losange a des angles de **60°** et **120°**
- Entre chaque losange, tu tournes de **60°** (360° / 6)

Consigne

1. Commence par l'étoile à 5 branches — c'est plus simple !
2. Ensuite, essaie l'étoile à 6 branches avec les losanges — c'est plus complexe car tu dois dessiner 6 formes distinctes.

🎨 Défi Final : Dessin Géométrique Libre

Difficulté : ⭐⭐⭐⭐ Expert

Objectif

Réaliser le dessin de **ton choix** en utilisant des formes géométriques.

Contrainte obligatoire

Ton dessin doit contenir **au moins 4 triangles ou quadrilatères différents** parmi ceux-ci :

- Triangle équilatéral
- Triangle isocèle
- Triangle rectangle
- Carré
- Rectangle
- Losange
- Parallélogramme
- Trapèze isocèle
- Trapèze rectangle

Idées de dessins

- 🏠 Une maison
- 🌸 Une fleur
- ⭐ Une étoile complexe
- ❤️ Un cœur géométrique
- 🐱 Un animal (chat, chien, oiseau...)
- 🎨 Une œuvre abstraite

Conseil méthodologique

1. **Fais d'abord un croquis sur papier** 📝
2. Planifie tes mesures (longueurs des côtés)
3. Planifie tes angles
4. Note les coordonnées de départ de chaque forme
5. Ensuite seulement, code dans Scratch !

Ce qu'on évalue

- ✅ Respect de la contrainte (4+ formes différentes)
- ✅ Qualité du code (utilisation de boucles, pas de répétition inutile)
- ✅ Créativité du dessin
- ✅ Précision des formes (angles et longueurs corrects)

💡 Astuces générales

L'astuce des 360°

La somme des angles extérieurs d'un polygone = **toujours 360°**

Donc pour un polygone à N côtés :

- Angle de rotation = 360° / N

Forme	Côtés	Angle de rotation
Triangle	3	360° / 3 = **120°**
Carré	4	360° / 4 = **90°**
Pentagone	5	360° / 5 = **72°**
Hexagone	6	360° / 6 = **60°**

Debug ton dessin

Si ta forme ne se ferme pas correctement :

1. Vérifie que la somme de tes rotations = 360°
2. Vérifie que tu as le bon nombre de répétitions
3. Vérifie que tu tournes dans le bon sens

Bon courage et amuse-toi bien ! 🎮✨

CHAPITRE 2

Introduction à Lego Mindstorm EV3 — Quand LEGO devient sérieux 🤖🧱

> 🎯 ***Objectif du module*** *: À la fin de ce module, tu seras capable de programmer un robot Lego EV3, comprendre les interactions capteurs/moteurs, et construire une base motrice fonctionnelle. Bienvenue dans le monde où tes briques d'enfance deviennent un outil d'ingénieur.*

Slides du chapitre

- lego mindstorm.pdf

🗺️ Où on en est dans le parcours ?

Avant de plonger dans les briques, situons-nous :

Partie	Contenu	Status
Programmation visuelle	Algobot, Scratch	✅ Fait
Robotique physique	**Lego Mindstorm EV3**	**👉 ON EST LÀ**
Électronique	Arduino, capteurs	🔜 Bientôt
Drones	Tello, Python	🔜 Plus tard

Pourquoi Lego Mindstorm maintenant ? Parce que c'est le pont parfait entre la programmation visuelle (Scratch) et l'électronique réelle (Arduino). On garde la programmation par blocs, mais on contrôle un VRAI robot avec de VRAIS moteurs et capteurs. C'est comme passer du simulateur de conduite à une vraie voiture... mais en moins dangereux. 🚗➡️🤖

🤔 C'est quoi Lego Mindstorm EV3 ?

Le concept

Lego Mindstorm, c'est le kit robotique éducatif de Lego lancé en 1998 et devenu une référence mondiale dans l'enseignement de la robotique. La version **EV3** (Evolution 3, sortie en 2013) est la plus aboutie de la gamme "classique".

Les composants clés

Composant	Rôle	Équivalent Arduino
Brique EV3	Le cerveau — microcontrôleur + écran + boutons	Arduino + shield LCD
Moteurs Large	Mouvement puissant, encodeur intégré	Moteur DC + encodeur
Moteurs Medium	Mouvement précis, plus compact	Servo moteur
Capteur tactile	Détecte le contact physique	Bouton poussoir
Capteur couleur	Détecte couleurs et intensité lumineuse	Capteur RGB ou LDR
Capteur ultrason	Mesure les distances	HC-SR04
Capteur gyroscope	Mesure l'orientation et rotation	MPU6050

Pourquoi c'est génial pour apprendre ?

1. **Pas de soudure, pas de câblage foireux** — tout se clipse
2. **Feedback immédiat** — l'écran de la brique affiche les infos
3. **Programmation visuelle** — même logique que Scratch
4. **Robuste** — tu peux (presque) pas le casser
5. **Documenté** — des tonnes de ressources officielles et communautaires

> 💡 ***Fun fact** : Le nom "Mindstorm" vient du livre "Mindstorms: Children, Computers, and Powerful Ideas" de Seymour Papert (1980), un pionnier de l'éducation par la robotique. LEGO a collaboré avec le MIT Media Lab pour créer ce produit. Oui, c'est du sérieux.*

⚠️ Note importante : Fin de vie du produit

Soyons transparents : LEGO a arrêté la production des Mindstorm EV3 et l'application **sera disponible jusqu'au 31 juillet 2026**.

Ce que ça veut dire pour nous :

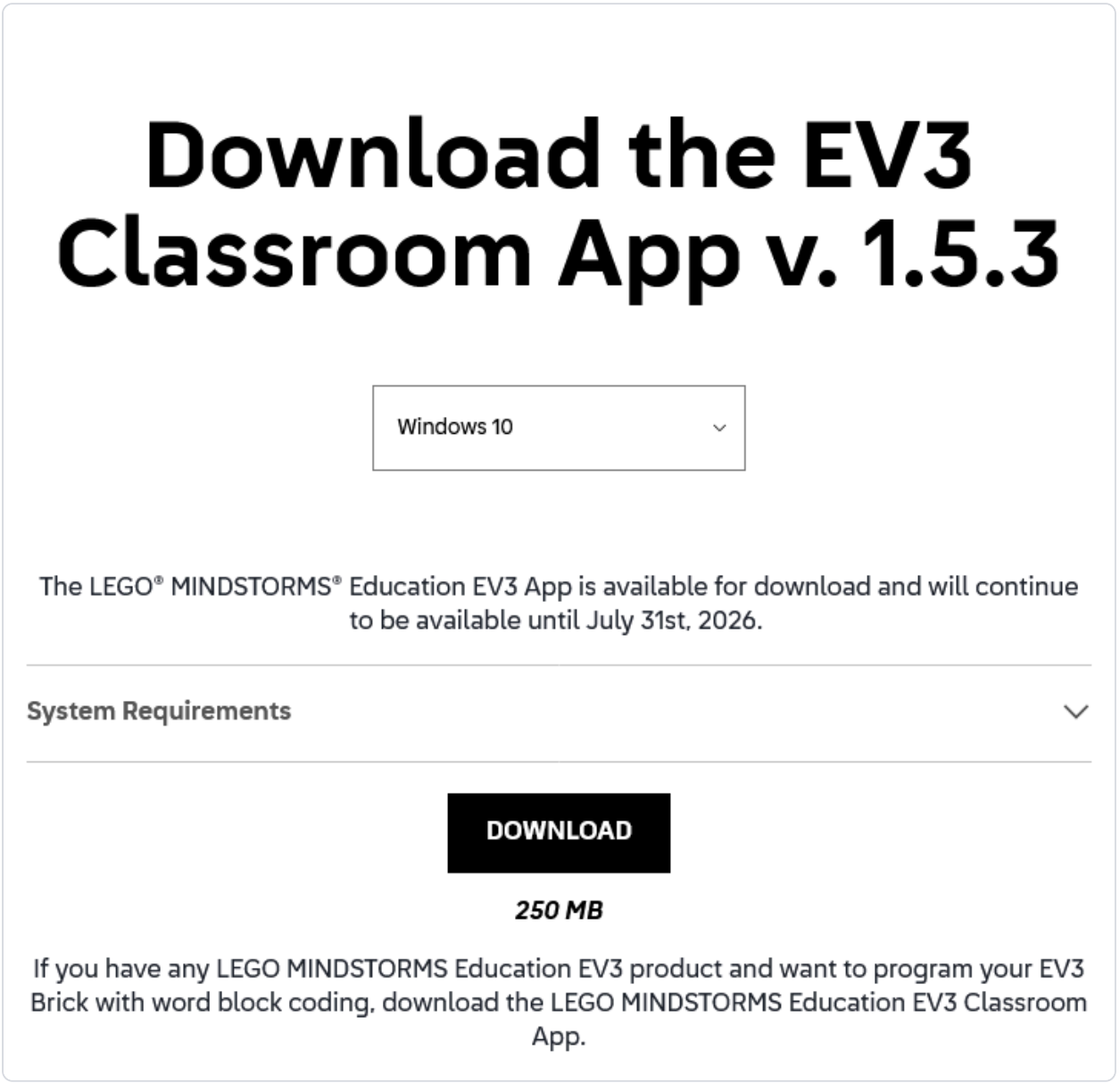
- ✅ On a largement le temps de l'utiliser cette année
- ✅ Le logiciel reste téléchargeable et fonctionnel
- ✅ Les kits physiques fonctionneront toujours après 2026
- ⚠️ Plus de mises à jour du logiciel après cette date

C'est comme avoir une PS4 : Sony ne la fabrique plus, mais elle marche toujours très bien pour jouer. 🎮

📥 Étape 1 : Installation du logiciel

Téléchargement

1. Va sur education.lego.com/en-us/downloads/mindstorms-ev3/software
2. Sélectionne **Windows 10** dans le menu déroulant
3. Clique sur **DOWNLOAD** (250 MB environ)
4. Installe le logiciel (suivant, suivant, suivant... tu connais la chanson)

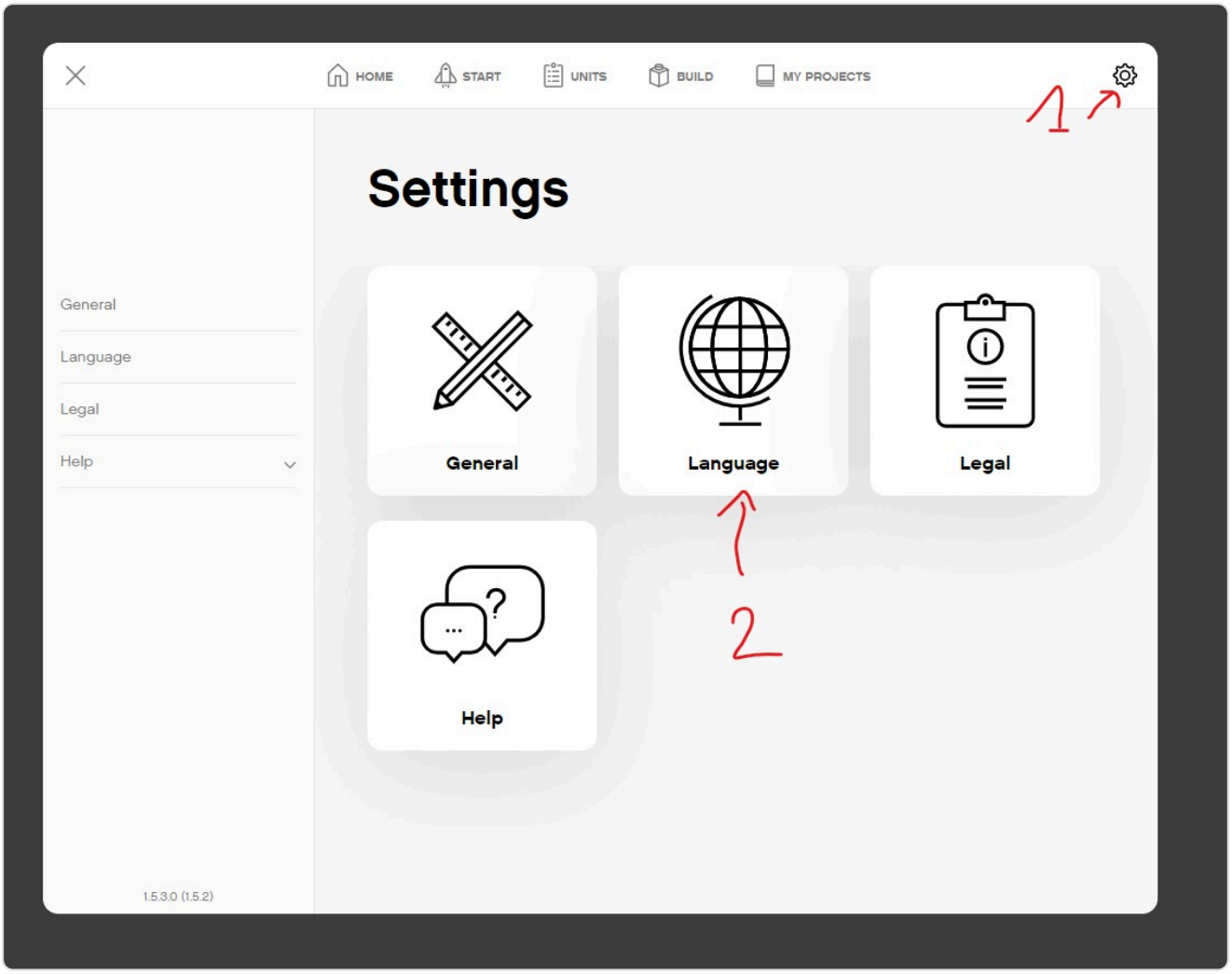


Configuration de la langue

Une fois le logiciel lancé :

1. Clique sur l'icône ⚙️ **Paramètres** (en haut à droite)
2. Sélectionne **Language**

3. Choisis **Français**
4. Redémarre l'application si nécessaire



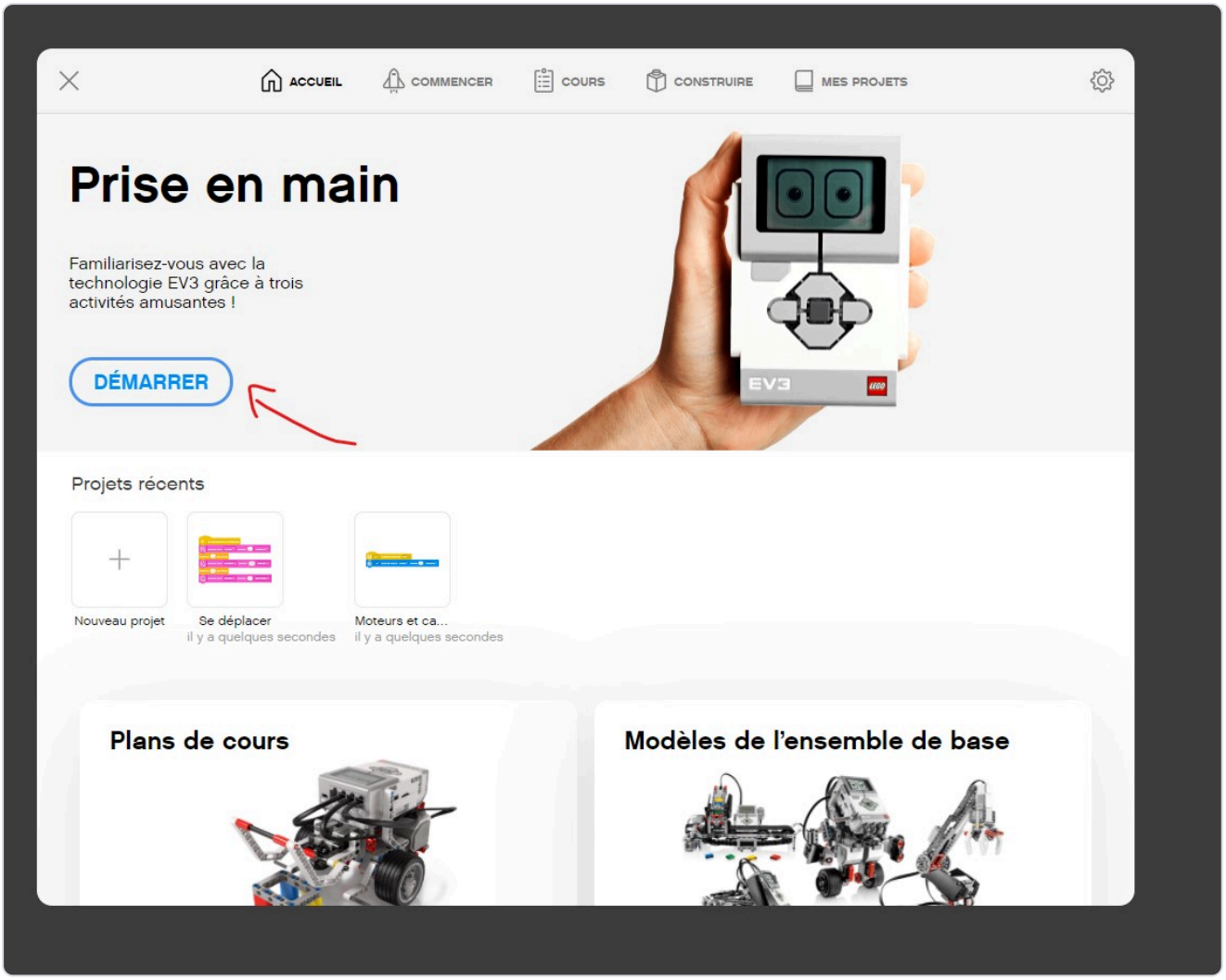
> 💡 ***Astuce*** *: L'interface en français rendra les instructions plus claires pour tout le monde. Pas de « Motor Large Port B » mais « Moteur large Port B ».*

🚀 Étape 2 : Prise en main — Les 3 activités de base

Avant de construire des robots complexes, on va faire les **3 activités de mise en route**. C'est comme un tutoriel de jeu vidéo : obligatoire mais rapide.

Comment y accéder ?

1. Sur l'écran d'accueil, clique sur **"DÉMARRER"** dans la section "Prise en main"
2. Ou va dans **COMMENCER** dans le menu du haut



Les 3 étapes

Étape	Nom	Ce qu'on apprend	Durée estimée
1️⃣	**Hello !**	Créer ton premier programme, faire parler la brique	10-15 min
2️⃣	**Moteurs et capteurs**	Contrôler les entrées et les sorties	15-20 min
3️⃣	**Se déplacer**	Construire une base motrice basique	20-30 min



Pourquoi c'est important ?

Ces 3 activités établissent les **fondations** :

- Tu apprends l'interface de programmation
- Tu comprends comment connecter la brique au PC
- Tu vois la relation entrée (capteur) → traitement (programme) → sortie (moteur)

C'est exactement le pattern qu'on retrouvera en Arduino : `digitalRead()` → logique → `digitalWrite()`. Même combat, emballage différent.

📚 Étape 3 : Télécharger les plans de cours

Le logiciel EV3 Classroom inclut des **plans de cours complets** avec instructions de montage, programmes exemples, et exercices guidés. C'est de l'or pédagogique.

Comment télécharger ?

1. Va dans l'onglet **COURS** dans le menu du haut

2. Tu verras plusieurs plans de cours disponibles
3. Clique sur chaque plan pour voir l'icône de **téléchargement** (flèche vers le bas)
4. Télécharge-les tous (ça prend quelques minutes)



Les 2 plans qu'on va suivre

Plan de cours	Thème	Ce qu'on va faire
Robot formateur	Robotique, Ingénierie, Informatique, STIAM	Configurer la base motrice et ses extensions
Laboratoire d'ingénierie	Ingénierie, Sciences, STIAM	Construire des machines et mécanismes



> ⚠️ ***Ordre important*** *: On commence par* ***Robot formateur*** *puis* ***Laboratoire d'ingénierie****. Le premier pose les bases, le second approfondit.*

👥 Organisation des équipes et du matériel

Le contexte

On dispose de **7 kits Lego Mindstorm EV3** plus ou moins complets.

Le hic : les pièces ont été mélangées au fil du temps. Un tri a été fait mais il faudra peut-être se répartir certaines pièces entre équipes.

Répartition des équipes

Avec **18 étudiants** présents, voici les options :

Option	Configuration	Avantages	Inconvénients
A	6 équipes de 3	Tout le monde en groupe égal, 1 kit spare	Groupes peut-être trop grands
B	**4 équipes de 3 + 3 équipes de 2**	**Utilise tous les kits, flexibilité**	Groupes inégaux

Recommandation : Option B (4 équipes de 3 + 3 équipes de 2) — on utilise tous les kits disponibles et les petits groupes de 2 peuvent avancer plus vite.

🤝 Règles de vie pour le matériel

1. **Pas de drama pour les pièces** — si une équipe a des pièces en trop, on partage équitablement
2. **Inventaire en début et fin de séance** — on vérifie qu'on n'a rien perdu
3. **Manipulation soignée** — c'est pas parce que c'est du LEGO qu'on peut le balancer
4. **Signaler les pièces manquantes** — plutôt que de galérer en silence

> 💡 ***Pro tip*** *: Désignez un "responsable matos" par équipe. Cette personne vérifie que tout est rangé en fin de séance. Ça évite de retrouver des engrenages dans les poches trois semaines plus tard.*

🎯 Objectifs pédagogiques du module

À la fin de ce module Lego Mindstorm, tu seras capable de :

Savoir (connaissances)

- ☑️ Identifier les différents composants d'un kit EV3 (brique, moteurs, capteurs)
- ☑️ Comprendre la logique entrée → traitement → sortie
- ☑️ Expliquer le rôle de chaque type de capteur

Savoir-faire (compétences)

- ☑️ Programmer la brique EV3 avec l'interface par blocs
- ☑️ Construire une base motrice fonctionnelle
- ☑️ Utiliser les capteurs pour créer des comportements (évitement d'obstacle, suivi de ligne)
- ☑️ Débugger un programme qui ne fonctionne pas

Savoir-être (attitudes)

- ☑️ Travailler en équipe sur un projet technique
- ☑️ Gérer le matériel de manière responsable

- ☑️ Documenter et partager ses solutions

✅ Checklist de démarrage

Avant de commencer le premier exercice, vérifie que :

- [ ] Le logiciel EV3 Classroom est installé
- [ ] La langue est configurée en français
- [ ] Les plans de cours sont téléchargés
- [ ] Tu as accès à un kit EV3 avec ta équipe
- [ ] La brique EV3 est chargée (ou branchée)
- [ ] Tu as identifié les pièces de base (brique, 2 moteurs large, capteurs)

🔗 Lien entre Mindstorm et la suite du cours

Ce qu'on apprend avec Lego Mindstorm va directement servir pour Arduino :

Concept Mindstorm	Équivalent Arduino
Bloc "Moteur Large"	`analogWrite()` + driver L298N
Bloc "Capteur Ultrason"	Librairie `NewPing` + HC-SR04
Bloc "Attendre"	`delay()`
Bloc "Boucle"	`loop()` + `while()`
Bloc "Si... Alors"	`if()`
Port d'entrée (1-4)	Pins analogiques/digitales
Port de sortie (A-D)	Pins PWM

La différence ? Avec Arduino, on tape du code au lieu de glisser des blocs. Mais la LOGIQUE est identique. Si tu comprends Mindstorm, tu comprendras Arduino.

🆘 En cas de problème

La brique ne se connecte pas au PC

1. Vérifie le câble USB
2. Vérifie que la brique est allumée
3. Redémarre le logiciel
4. Essaie un autre port USB

Le programme ne fait rien

1. Vérifie que tu as bien cliqué sur "Télécharger et exécuter"
2. Vérifie les ports (le moteur est-il sur le bon port ?)
3. Vérifie les piles de la brique

Il manque des pièces pour le montage

1. Demande aux autres équipes si elles ont des surplus
2. Vérifie le 7ème kit (spare)
3. Adapte le montage si possible (LEGO c'est flexible!)

📖 Ressources complémentaires

- **Documentation officielle** : Incluse dans l'application EV3 Classroom
- **EV3Lessons** : ev3lessons.com — Tutoriels communautaires, challenges, ressources pédagogiques
- **LEGO Education** : education.lego.com/en-us/lessons — Leçons officielles et plans de cours
- **Robot Square** : robotsquare.com — Projets et tutoriels par Laurens Valk (auteur du livre officiel)
- **YouTube** : Cherche "EV3 tutorial français" ou "LEGO Mindstorm EV3 débutant"

➡️ Prochaine étape

Une fois le logiciel installé et les plans de cours téléchargés, on attaque la **Mise en route** ensemble en classe. Prépare-toi à faire parler ton premier robot ! 🗣️🤖

Module rédigé pour le cours Drone & Robotique — IFOSUP Wavre "De la brique au drone, un pas à la fois" 🧱➡️🚁

2.1 Premiers Pas avec le Robot EV3

> 🎯 ***Objectif du chapitre*** *: À la fin de ce chapitre, tu seras capable de programmer ton robot EV3 pour qu'il se déplace avec précision, détecte des obstacles, et saisisse des objets. Les bases pour créer un robot vraiment autonome !*

Slides du chapitre

- Premiers pas robot EV3.pdf

🗺️ Ce qu'on va apprendre aujourd'hui

Leçon	Titre	Tu sauras faire...
1	Déplacements et rotations	Faire avancer le robot d'une distance précise
2	Objets et obstacles	Détecter un obstacle et réagir
3	Saisir et libérer	Utiliser un bras motorisé pour déplacer des objets

Temps estimé : 2h30 à 3h de pratique

🧠 Avant de Commencer : Comprendre son Robot

Avant de se lancer dans la programmation, prenons 5 minutes pour comprendre comment notre robot "réfléchit".

Les Moteurs Encodés : Le Secret de la Précision

Tu t'es déjà demandé comment un robot sait qu'il a avancé de 50 cm exactement ? Il n'a pas d'yeux pour regarder le sol...

La réponse : **les moteurs encodés**.

Un moteur encodé, c'est un moteur normal avec un petit bonus : un **compteur intégré** qui compte chaque fraction de tour. C'est exactement comme le compteur kilométrique d'une voiture !

🚗 Dans une voiture	🤖 Dans le robot EV3
Compteur kilométrique	Encodeur du moteur
Compte les km parcourus	Compte les tours de roue
Affiche sur le tableau de bord	Renvoie l'info au programme

Trois Façons de Contrôler un Déplacement

Dans l'application EV3, tu peux dire au robot de bouger de trois manières différentes :

Méthode	Ce que tu dis au robot	Précision	Quand l'utiliser
⏱️ **Secondes**	"Avance pendant 2 secondes"	⭐ Faible	Tests rapides
📐 **Degrés**	"Tourne tes moteurs de 360°"	⭐⭐ Moyenne	Virages précis
🔄 **Rotations**	"Fais 3 tours de roue"	⭐⭐⭐ Haute	Distances exactes

💡 Pourquoi les secondes sont imprécises ?

Parce que la vitesse du robot dépend de l'état des piles ! Avec des piles neuves, "2 secondes" = plus de distance qu'avec des piles faibles. Les rotations, elles, comptent les tours de roue : 3 tours = 3 tours, peu importe les piles.

Le Capteur Ultrason : Voir Sans les Yeux

Le capteur ultrason permet au robot de "voir" les obstacles devant lui. Mais comment ça marche ?

C'est le même principe que l'**écholocation des chauves-souris** ou le **sonar des sous-marins** :



Le capteur émet un son très aigu (inaudible pour nous) et mesure le temps que met l'écho à revenir. Plus c'est long, plus l'obstacle est loin !

Ce que le capteur peut détecter :

✅ Fonctionne bien	⚠️ Difficile	❌ Ne fonctionne pas
Murs	Surfaces molles (tissus)	Verre transparent
Boîtes en carton	Objets très petits	Miroirs (angle)
Ta main	Surfaces très inclinées	Trous / vide

Portée : de 3 cm à 250 cm (2,5 mètres)

Le Pseudocode : Réfléchir Avant de Coder

Avant de construire une maison, on fait un plan. Avant de coder... on fait du **pseudocode** !

Le pseudocode, c'est écrire son programme **en français** avant de le traduire en blocs. Ça évite de cliquer au hasard et de se perdre.

Exemple pour un robot qui évite les obstacles :

```
DÉBUT DU PROGRAMME

    RÉPÉTER indéfiniment :

        Avancer tout droit

        SI la distance devant < 20 cm ALORS
            S'arrêter
            Reculer un peu
            Tourner à droite
        FIN SI

    FIN RÉPÉTER

FIN DU PROGRAMME
```

Une fois que tu as écrit ça, traduire en blocs EV3 devient beaucoup plus facile !

📝 Leçon 1 : Déplacements et Rotations

Prérequis

Tu dois avoir **construit ta base motrice** (train roulant) lors de la séance précédente. Si ce n'est pas fait, suis les instructions de montage "Se déplacer" dans l'application avant de continuer.

Objectif

Apprendre à programmer des déplacements **précis et contrôlés** avec la base motrice.

Le Défi Final

Faire parcourir exactement **84 cm** à ton robot. Ni plus, ni moins !

Les Types de Virages

Ton robot peut tourner de plusieurs façons :

Type de virage	Comment ça marche	Quand l'utiliser
Pivot	Une roue avance, l'autre recule	Tourner sur place (demi-tour)

Type de virage	Comment ça marche	Quand l'utiliser
Swing	Une roue avance, l'autre reste fixe	Virage moyen
Arc	Les deux roues avancent, mais à vitesses différentes	Virage doux en avançant

Exercice Pratique

1. **Ouvre** l'application EV3 Classroom
2. **Connecte** ton robot via Bluetooth ou USB
3. **Teste** les piles de programmation fournies
4. **Observe** : quel bloc fait avancer ? tourner ?
5. **Mesure** : place ton robot, lance le programme, mesure la distance parcourue

🧮 Trouver Comment Calculer la Distance

Pour atteindre le niveau Diamant (parcourir 84 cm avec précision), il va falloir réfléchir un peu ! Voici quelques questions pour te guider :

Étape 1 : Observer ta roue

- Qu'est-ce qu'une roue, géométriquement parlant ?
- Quand ta roue fait **un tour complet**, quelle distance parcourt-elle au sol ?
- Comment appelle-t-on la "longueur du pourtour" d'un cercle ?

Étape 2 : Se souvenir des maths

- Tu as sûrement appris au collège comment calculer cette fameuse longueur...
- Ça implique le diamètre (ou le rayon) et un nombre très célèbre qui commence par 3,14...
- Un indice : cherche "circonférence d'un cercle" si tu as oublié !

Étape 3 : Mesurer

- Prends ta roue EV3 et mesure son diamètre avec une règle
- Applique ta formule pour trouver la distance parcourue en 1 rotation

Étape 4 : Calculer

- Si tu sais quelle distance fait 1 rotation, comment trouver le nombre de rotations pour 84 cm ?

💡 **Astuce** : Tu peux aussi procéder par expérimentation ! Programme 1 rotation, mesure, et déduis le reste.

🎯 Auto-évaluation

Niveau	J'ai réussi à...
🥉 Bronze	Faire avancer le robot en ligne droite
🥈 Argent	Faire différents types de virages
🥇 Or	Créer un programme pour parcourir 84 cm
💎 Diamant	Utiliser les maths pour calculer la distance exacte

📝 Leçon 2 : Objets et Obstacles

Objectif

Utiliser le **capteur ultrason** pour détecter des obstacles et faire réagir le robot.

Construction Préalable

Avant de programmer, tu dois :

1. **Construire** l'extension capteur ultrason (instructions dans l'app)
2. **Connecter** le capteur sur le port **4**
3. **Construire** le cuboïde (l'obstacle à détecter)

Le Défi Final

Programmer le robot pour qu'il émette des **bips de plus en plus rapides** quand il s'approche d'un obstacle — exactement comme le radar de recul d'une voiture !

Comment Utiliser le Capteur

Le capteur renvoie un nombre : la **distance en centimètres**. Tu peux utiliser ce nombre dans des conditions :

```
SI distance < 30 cm ALORS
    Ralentir
FIN SI

SI distance < 10 cm ALORS
    S'arrêter complètement
    Jouer un son d'alerte
FIN SI
```

Exercice Pratique

1. **Place** le cuboïde à environ 50 cm devant le robot
2. **Programme** : le robot avance jusqu'à détecter l'obstacle
3. **Ajoute** : il s'arrête à 10 cm de l'obstacle
4. **Améliore** : il joue un son quand il s'arrête

🎯 Auto-évaluation

Niveau	J'ai réussi à...
🥉 Bronze	Arrêter le robot devant un obstacle
🥈 Argent	Afficher la distance sur l'écran du robot
🥇 Or	Faire varier la fréquence des bips selon la distance
💎 Diamant	Créer plusieurs zones (loin = lent, proche = rapide)

📝 Leçon 3 : Saisir et Libérer

Objectif

Utiliser un **outil motorisé** pour capturer et déplacer des objets.

Construction Préalable

Tu dois ajouter le **moteur moyen** avec son extension :

1. **Construire** l'extension moteur moyen (instructions dans l'app)
2. **Connecter** le moteur sur le port **A**
3. **Garder** le capteur ultrason branché (port 4)

Comment Fonctionne le Mécanisme ?

Ce n'est **pas une pince** qui attrape ! C'est plutôt un **enclos mobile** — un petit cadre qui peut monter et descendre :

1. Le cadre **descend** autour du cuboïde (comme une cage qui se referme)
2. Le cube est maintenant **prisonnier** dans l'enclos
3. Le robot **recule** → le cube glisse avec lui
4. Arrivé à destination, le cadre **remonte** → le cube est libéré !

C'est le même principe qu'un chariot élévateur simplifié.

Contrôler le Moteur Moyen

Rotation	Ce qui se passe
Positive (+)	Le cadre **descend** → capture le cube
Négative (-)	Le cadre **remonte** → libère le cube

⚠️ **Attention** : Ne force jamais le moteur en butée ! Si le cadre est déjà tout en bas, ne lui demande pas de descendre encore — ça peut abîmer le mécanisme. Limite le nombre de degrés.

Le Défi Final

Mission complète :

1. Le robot **avance** vers le cuboïde
2. Il **détecte** l'objet avec l'ultrason
3. Il **s'arrête** à bonne distance (le cube doit être sous le cadre)
4. Le cadre **descend** autour du cube
5. Le robot **recule** → le cube glisse avec lui
6. Arrivé au point de départ, le cadre **remonte**
7. Le cube est **libéré** !

Exercice Pratique

1. **Teste** d'abord le cadre tout seul : monter, descendre (combien de degrés ?)
2. **Vérifie** que le cadre descend assez bas pour entourer le cube
3. **Combine** : avancer + détecter + s'arrêter + descendre cadre
4. **Programme** le retour : reculer + remonter cadre
5. **Ajuste** les distances et timings si le cube s'échappe en route

🎯 Auto-évaluation

Niveau	J'ai réussi à...
🥉 Bronze	Faire monter et descendre le cadre au bon moment
🥈 Argent	Combiner la détection ultrason avec le mouvement du cadre
🥇 Or	Ramener le cuboïde au point de départ
💎 Diamant	Ajuster pour différentes distances de départ

🚀 Défis Bonus

Tu as fini les trois leçons ? Voici des défis supplémentaires !

Défi "L'Artiste" 🎨

Programme ton robot pour qu'il trace au sol la **première lettre de ton prénom**.

Indice : Décompose ta lettre en segments droits et en virages. Par exemple, un "L" = avancer + tourner 90° + avancer.

Défi "Le Parcours Complet" 🏁

Combine TOUT ce que tu as appris :

1. Le robot avance jusqu'à un obstacle
2. Il s'arrête à 10 cm
3. Il attrape l'objet
4. Il fait demi-tour (180°)
5. Il revient au départ
6. Il dépose l'objet

Bonus : Fais-le sans connaître à l'avance la position de l'objet !

📚 Vocabulaire à Retenir

Terme	Définition
Encodeur	Capteur intégré au moteur qui compte les tours
Rotation	Un tour complet de roue (360°)
Ultrason	Son très aigu, inaudible par l'humain
Capteur	Composant qui mesure quelque chose (distance, lumière...)
Actionneur	Composant qui fait une action (moteur, bras...)
Pseudocode	Programme écrit en langage naturel avant le vrai code

🔗 Pour Aller Plus Loin

- **Application officielle** : LEGO Education EV3 Classroom
- **Documentation** : education.lego.com
- **Idées de projets** : Les leçons 4, 5 et 6 du Robot Formateur (prochaines séances !)

2.2 Annexe : Le Paramètre Direction (Steering)

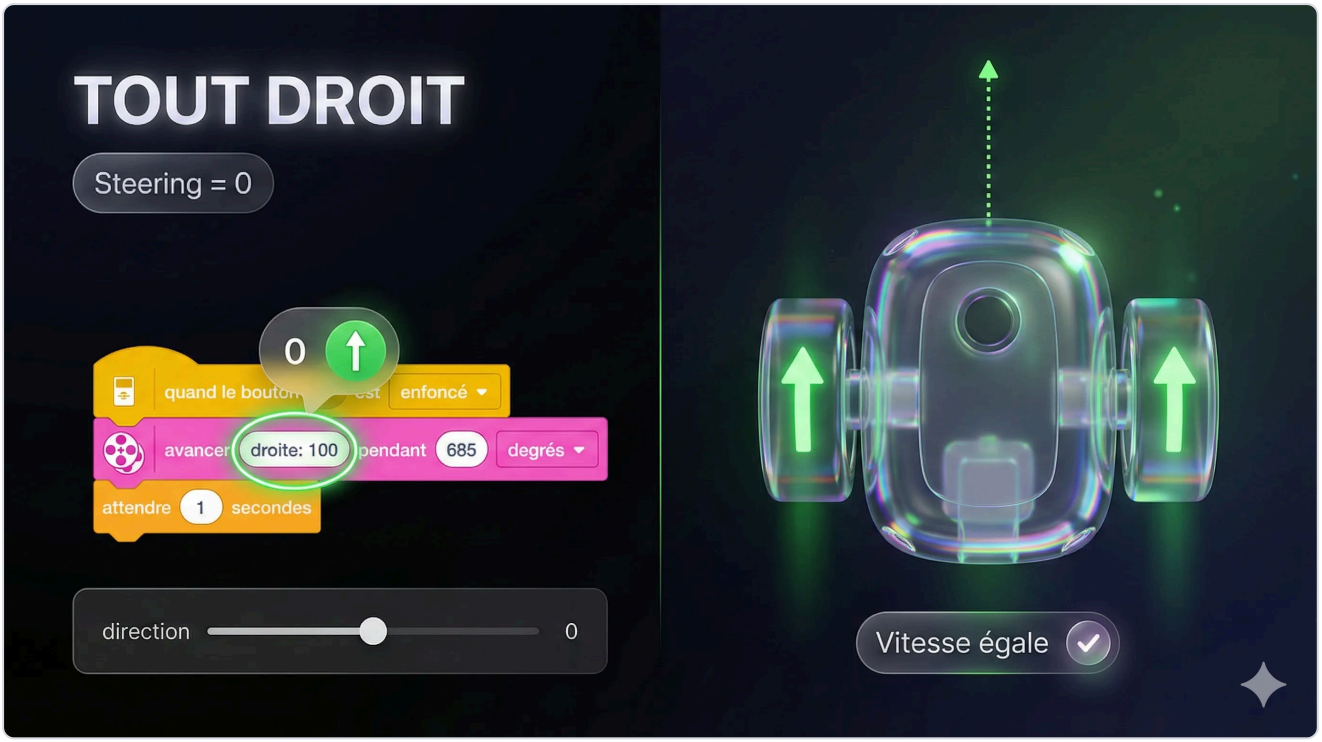
Le paramètre **direction** contrôle comment les deux roues tournent l'une par rapport à l'autre. Il va de **-100** (gauche) à **+100** (droite).

Vue d'ensemble



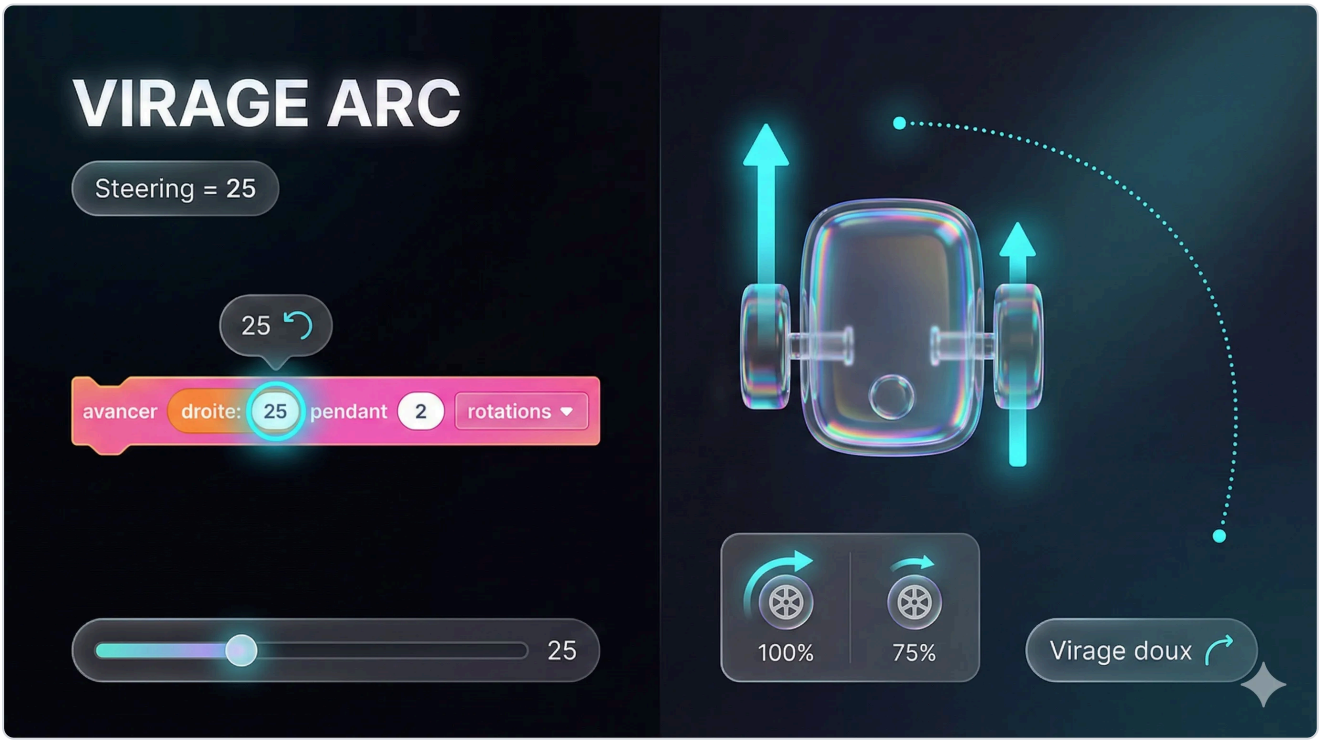
Tout Droit — Steering = 0

Les deux roues avancent à la même vitesse.



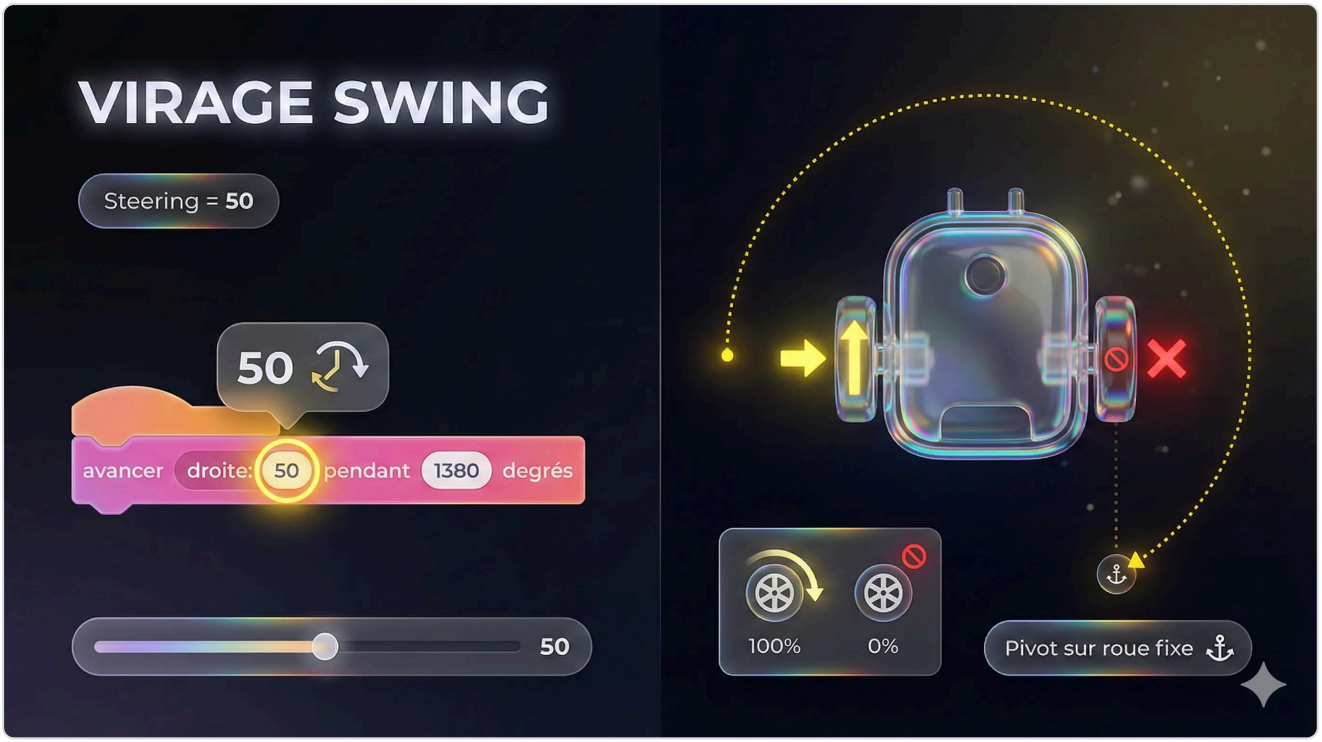
Virage Arc — Steering = ±25

Une roue va plus lentement → courbe douce.



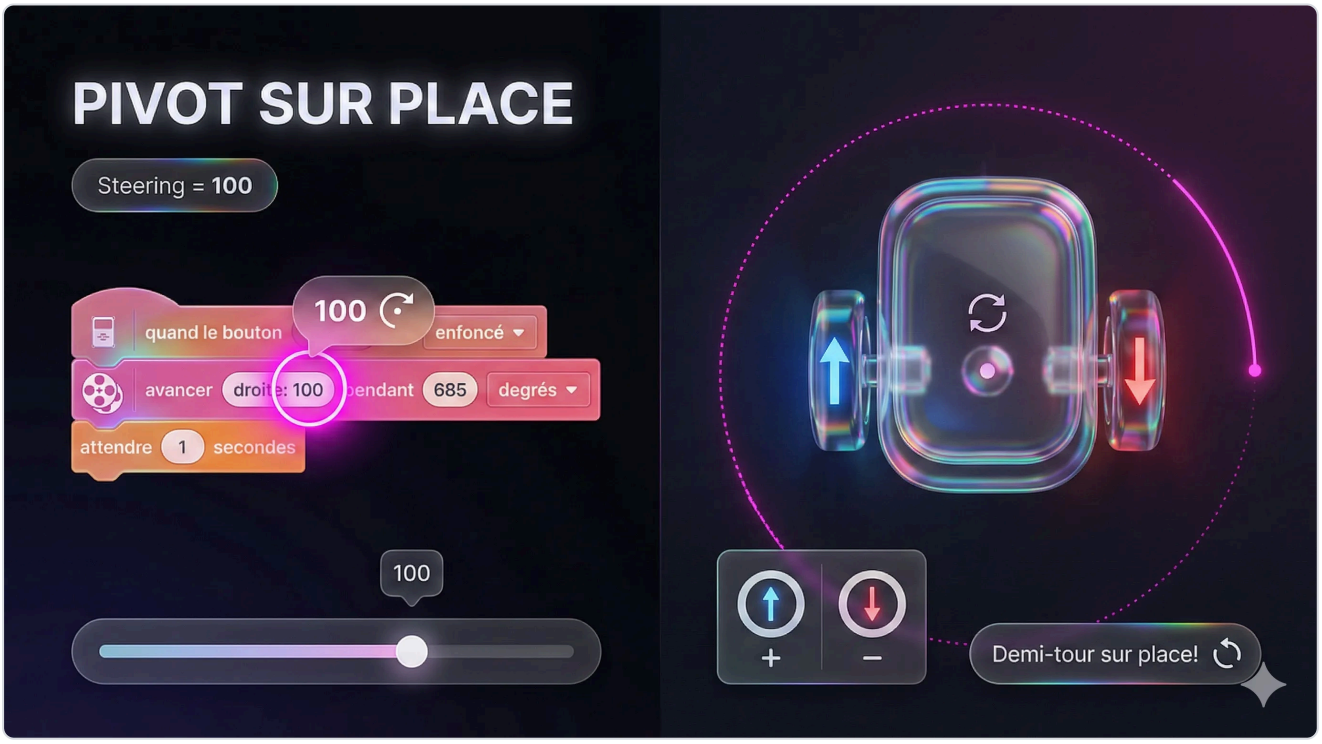
Virage Swing — Steering = ±50

Une roue s'arrête, l'autre avance → virage autour de la roue fixe.

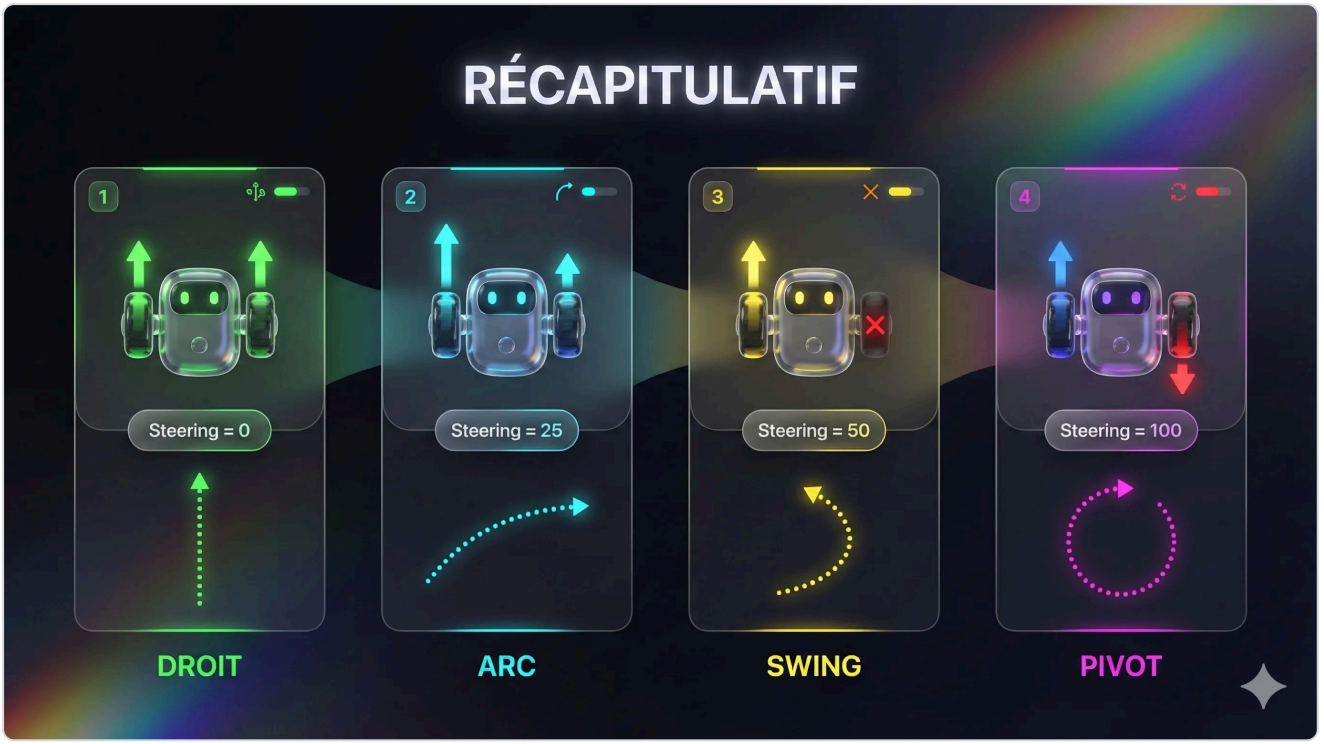


Pivot sur Place — Steering = ±100

Une roue avance, l'autre recule → rotation sur place.



Récapitulatif



Steering	Mode	Comportement
0	Droit	Deux roues égales
±25	Arc	Une roue ralentit
±50	Swing	Une roue fixe
±100	Pivot	Roues opposées

2.3 EV3 : Couleurs, Angles et Intégration

> 🎯 ***Objectif*** *: Maîtriser le capteur de couleur, le gyroscope, et combiner tous tes acquis dans un projet d'usine robotisée.*

Slides du chapitre

- ev3 suite.pdf

🗺️ Programme des Leçons 4, 5 et 6

Leçon	Titre	Nouveau capteur/concept	Tu sauras faire...
4	Couleurs et lignes	Capteur de couleur	Suivre une ligne noire au sol
5	Angles et motifs	Gyroscope + Mes Blocs	Tourner à des angles précis, créer des fonctions
6	Robot de l'usine	Intégration complète	Combiner tous les capteurs pour une mission

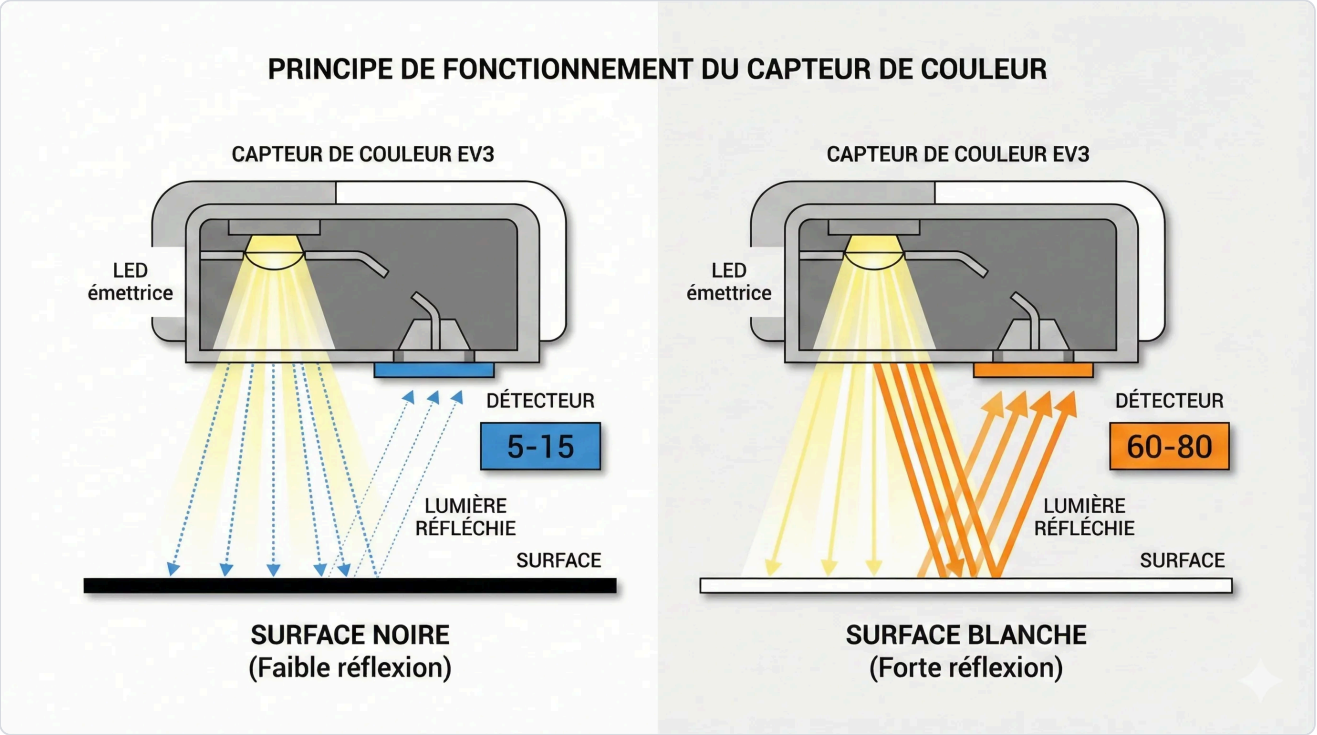
Temps estimé : 45-90 min par leçon (à ton rythme)

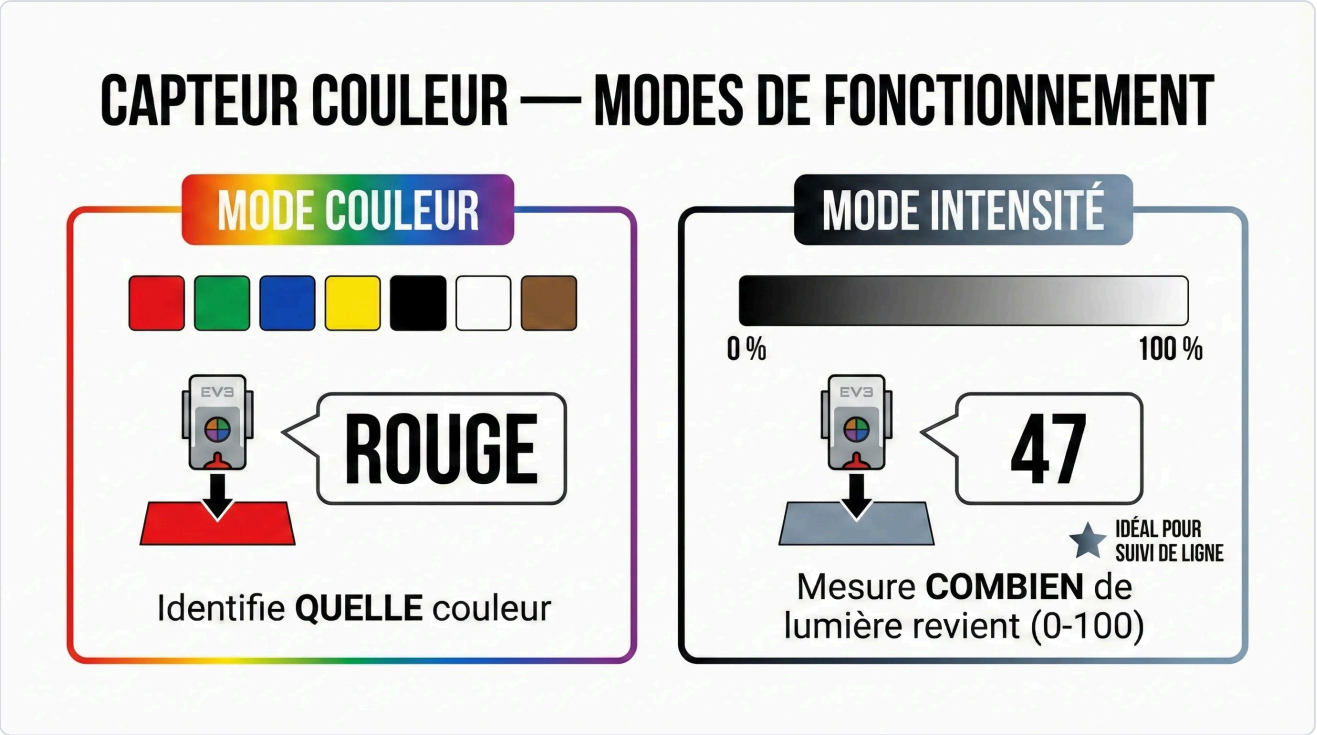
🎨 Leçon 4 : Le Capteur de Couleur

Comment ça marche ?

Le capteur de couleur émet une **lumière** et analyse ce qui **revient**. Deux modes principaux :

Mode	Ce qu'il mesure	Utilisation
Couleur	La couleur détectée (rouge, bleu, vert, jaune, noir, blanc, brun)	Trier des objets, détecter des zones
Intensité lumière réfléchie	Un nombre de 0 (noir) à 100 (blanc)	Suivre une ligne avec précision





💡 Le principe du suivi de ligne

Quand le capteur est au-dessus d'une **ligne noire** → valeur basse (ex: 5-15) Quand le capteur est au-dessus du **sol clair** → valeur haute (ex: 60-80)

Question pour réfléchir : Comment utiliser cette différence pour que le robot suive la ligne ?

Indice : le robot doit constamment corriger sa trajectoire en fonction de ce qu'il "voit".

⚠️ L'étalonnage : pourquoi c'est important ?

Les conditions de lumière changent ! Une salle éclairée au néon vs. lumière du jour → valeurs différentes.

Avant de programmer, fais lire au capteur :

1. La surface la plus **claire** (sol blanc/gris)
2. La surface la plus **foncée** (ligne noire)

Cela définit ta plage de référence.

🎯 Auto-évaluation Leçon 4

Niveau	J'ai réussi à...
🥉 Bronze	Arrêter la base motrice sur une ligne noire
🥈 Argent	Programmer le robot pour qu'il suive une ligne

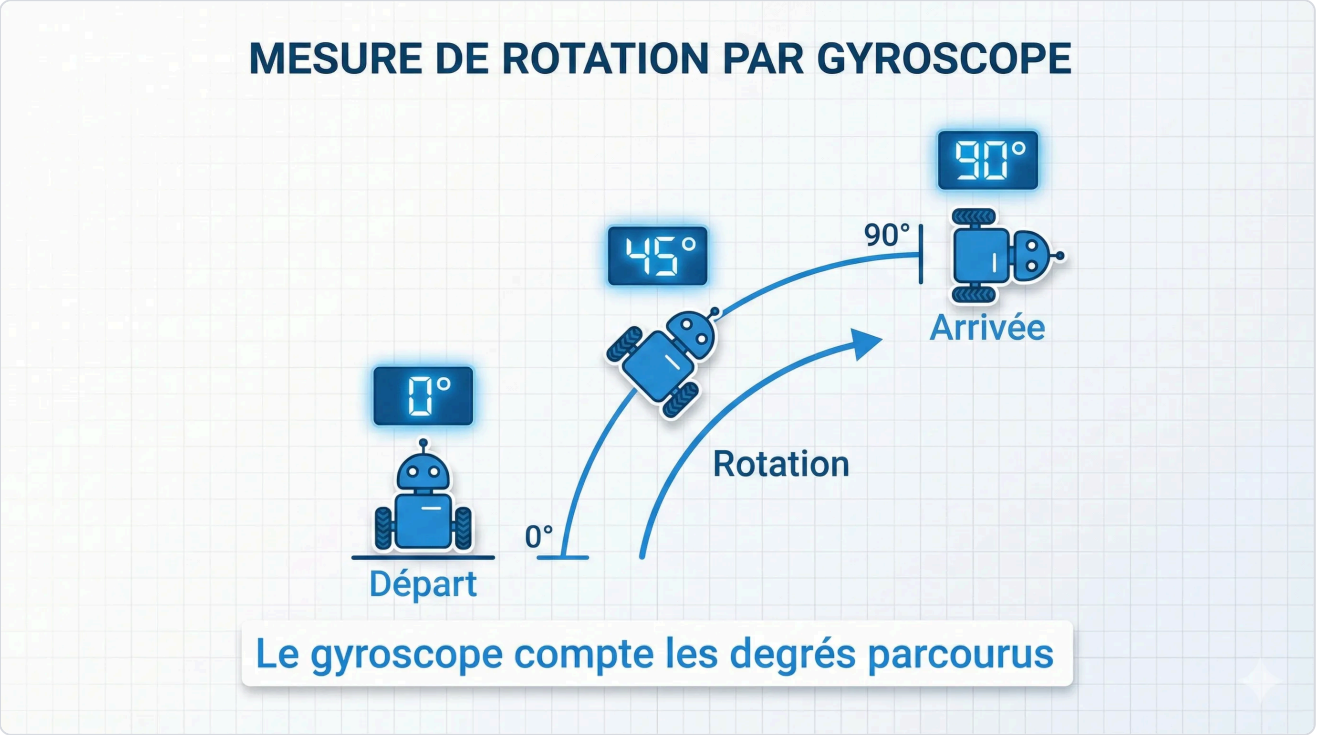
Niveau	J'ai réussi à...
🥇 Or	Suivre une ligne ET étalonner le capteur pour différentes lumières
💎 Diamant	Optimiser le programme pour suivre la ligne plus vite et plus efficacement

🧭 Leçon 5 : Le Capteur Gyroscopique

Comment ça marche ?

Le gyroscope mesure la **rotation** du robot sur lui-même. Il détecte les changements d'**orientation** (angle).

Ce qu'il mesure	Unité	Exemple
Angle de rotation	Degrés (°)	"J'ai tourné de 90°"
Vitesse de rotation	°/seconde	"Je tourne à 45°/s"



💡 Pourquoi c'est utile ?

Sans gyroscope, faire tourner le robot de **exactement 90°** est compliqué :

- Les roues peuvent glisser
- Le sol peut être irrégulier
- Les piles influencent la vitesse

Avec le gyroscope → le robot **sait** de combien il a tourné, peu importe les conditions !

⚠️ Le problème de la dérive (drift)

Le gyroscope doit être **parfaitement immobile** au démarrage. Sinon, il accumule des erreurs.

Si les valeurs bougent toutes seules (robot immobile) :

1. Débranche le capteur
2. Rebranche-le
3. Attends que le robot soit stable avant de lancer le programme

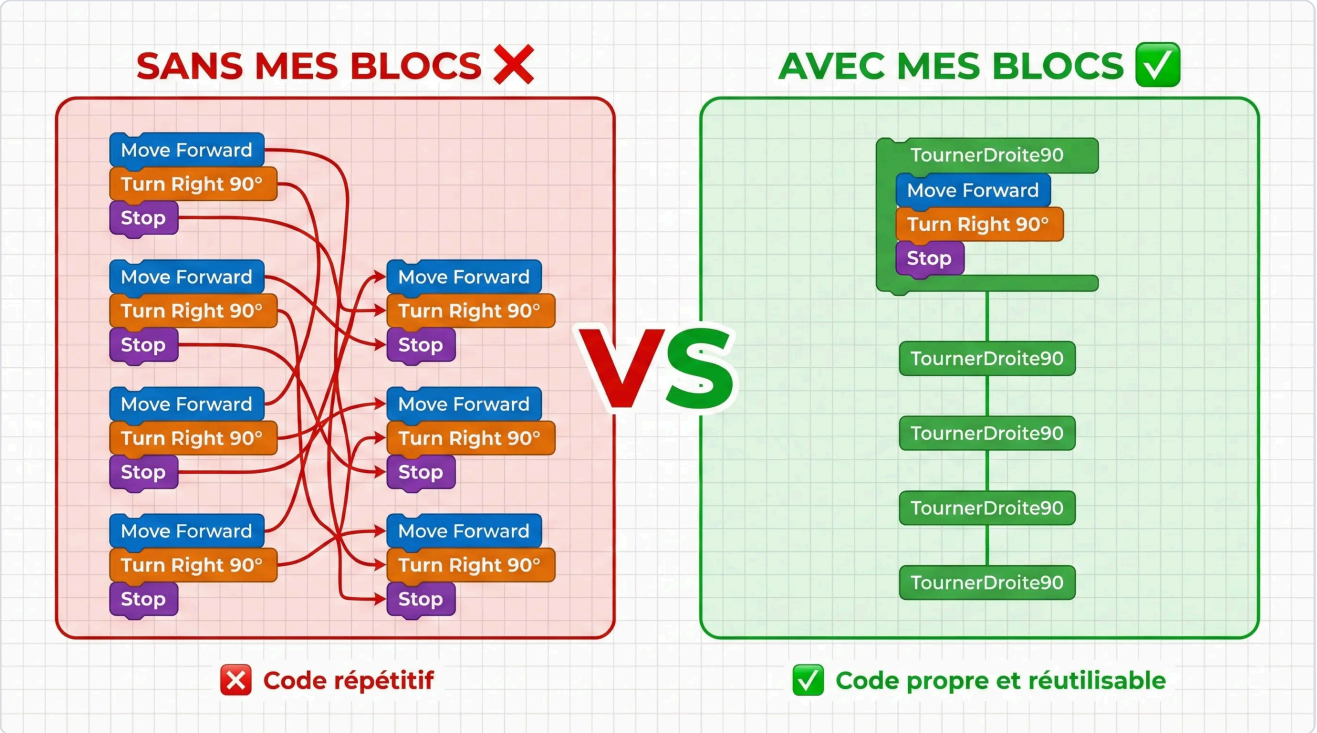
🧩 Nouveau concept : "Mes Blocs" (fonctions)

Tu vas répéter souvent les mêmes séquences (avancer, tourner 90°, avancer...). Au lieu de copier-coller, crée un **"Mes Blocs"** !

C'est comme une **recette nommée** que tu peux réutiliser :

```
Mon Bloc "Tourner90Droite" :
   → Activer moteurs en rotation opposée
   → Attendre que gyroscope = 90°
   → Arrêter moteurs
```

Ensuite, tu appelles juste "Tourner90Droite" dans ton programme. Plus propre, plus efficace !



🎯 Auto-évaluation Leçon 5

Niveau	J'ai réussi à...
🥉 Bronze	Faire tourner le robot de 45° avec le gyroscope
🥈 Argent	Créer un "Mes Blocs" pour faire un triangle
🥇 Or	Créer un "Mes Blocs" pour faire un carré précis
💎 Diamant	Utiliser mes blocs pour tracer plusieurs carrés consécutifs

🏭 Leçon 6 : Le Robot de l'Usine

Le contexte

Dans les vraies usines (comme celle de LEGO au Danemark !), des robots autonomes transportent des pièces entre les postes de travail. Ils suivent des lignes au sol et évitent les obstacles.

Ta mission

Ton robot doit accomplir **deux tâches** sur le parcours d'usine :

Tâche 1 — Transport d'objet :

- Partir de la zone de départ
- Saisir le cuboïde
- Le déposer au centre du cercle cible (zone de dépôt)

Tâche 2 — Suivi de ligne :

- Suivre la ligne noire (avec sa courbe)
- S'arrêter le plus près possible de l'objet volumineux
- Sans le toucher !

📐 Comprendre le parcours



Le parcours se compose de :

Élément	Description	Repère sur le schéma
Zone de départ	Là où ton robot commence	① ②
Cuboïde	L'objet à saisir et transporter	Petit carré près du départ
Cercle cible	Zone de dépôt pour le cuboïde	④ (cercle concentrique)
Ligne noire	Chemin à suivre (avec courbe)	⑤ ⑥
Objet volumineux	Obstacle de fin de ligne	⑦ (rectangle hachuré)

Dimensions indicatives :

- Parcours total : ~180 cm de large
- Distance départ → cuboïde : ~84 cm (tiens, ça te rappelle quelque chose ?)
- Hauteur du parcours : ~100 cm

💡 Stratégie suggérée

Pour cette mission complexe, tu auras besoin de **combiner** :

- Capteur ultrason → détecter les objets
- Capteur de couleur → suivre la ligne
- Capteur gyroscopique → tourner précisément vers la cible

- Moteur moyen → saisir/libérer le cuboïde

Conseil : Découpe le problème en sous-problèmes. Teste chaque partie séparément avant d'assembler !

🎯 Auto-évaluation Leçon 6

Niveau	J'ai réussi à...
🥉 Bronze	Accomplir l'une des deux tâches
🥈 Argent	Accomplir les deux tâches (même imparfaitement)
🥇 Or	Accomplir les deux tâches avec de bons résultats
💎 Diamant	Ajouter des fonctionnalités originales et efficaces

📚 Vocabulaire à Retenir

Terme	Définition
Intensité lumière réfléchie	Mesure de 0 (noir) à 100 (blanc) de la lumière qui revient au capteur
Étalonnage	Réglage du capteur pour s'adapter aux conditions de lumière
Gyroscope	Capteur qui mesure la rotation (angle et vitesse)
Dérive (drift)	Erreur qui s'accumule dans le gyroscope s'il n'est pas bien initialisé
Mes Blocs	Bloc personnalisé réutilisable (équivalent d'une fonction en programmation)

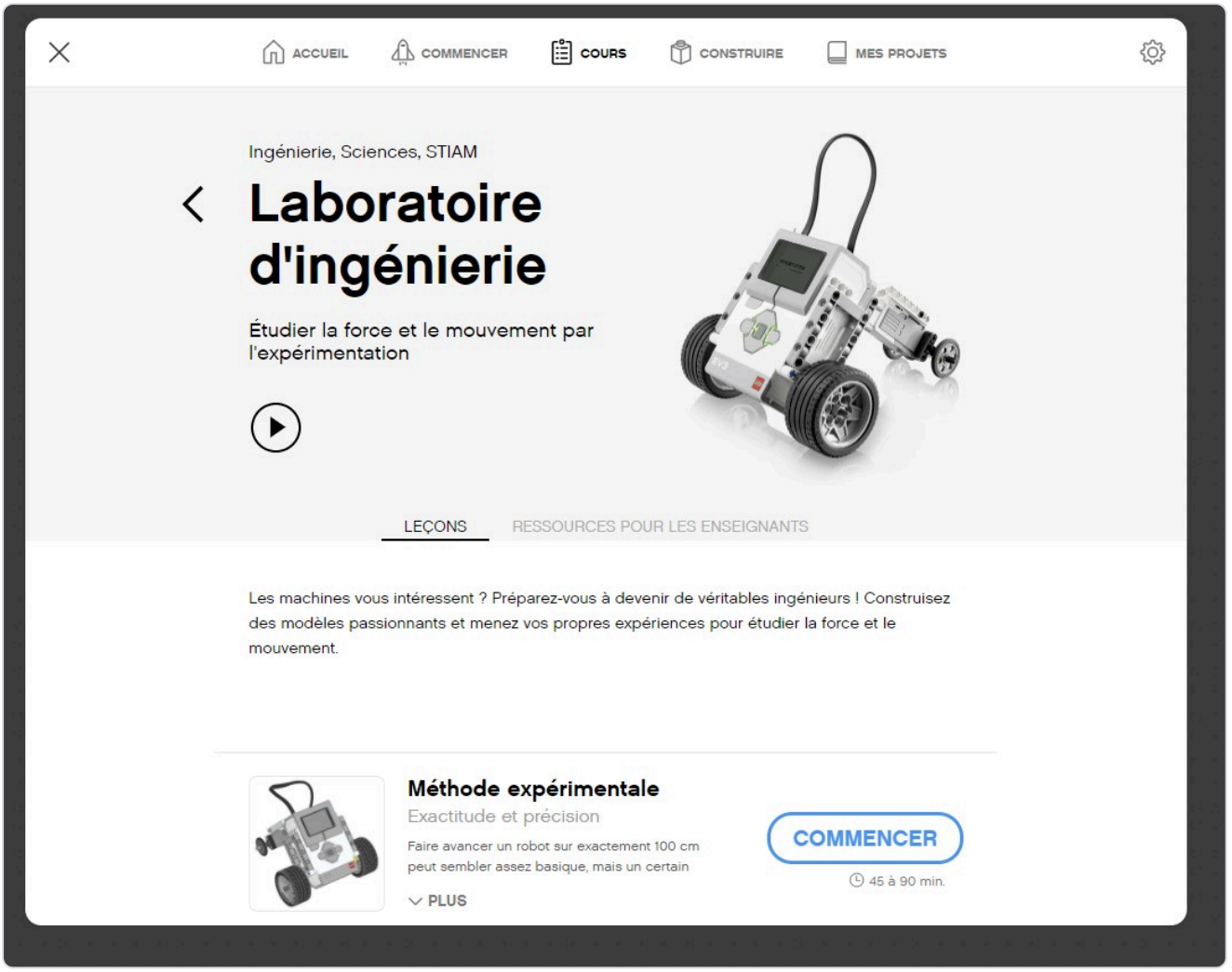
🚀 Pour Aller Plus Loin

Autres activités dans l'application EV3

Tu as terminé les 6 leçons du Robot Formateur ? Bravo ! L'application propose d'autres défis :

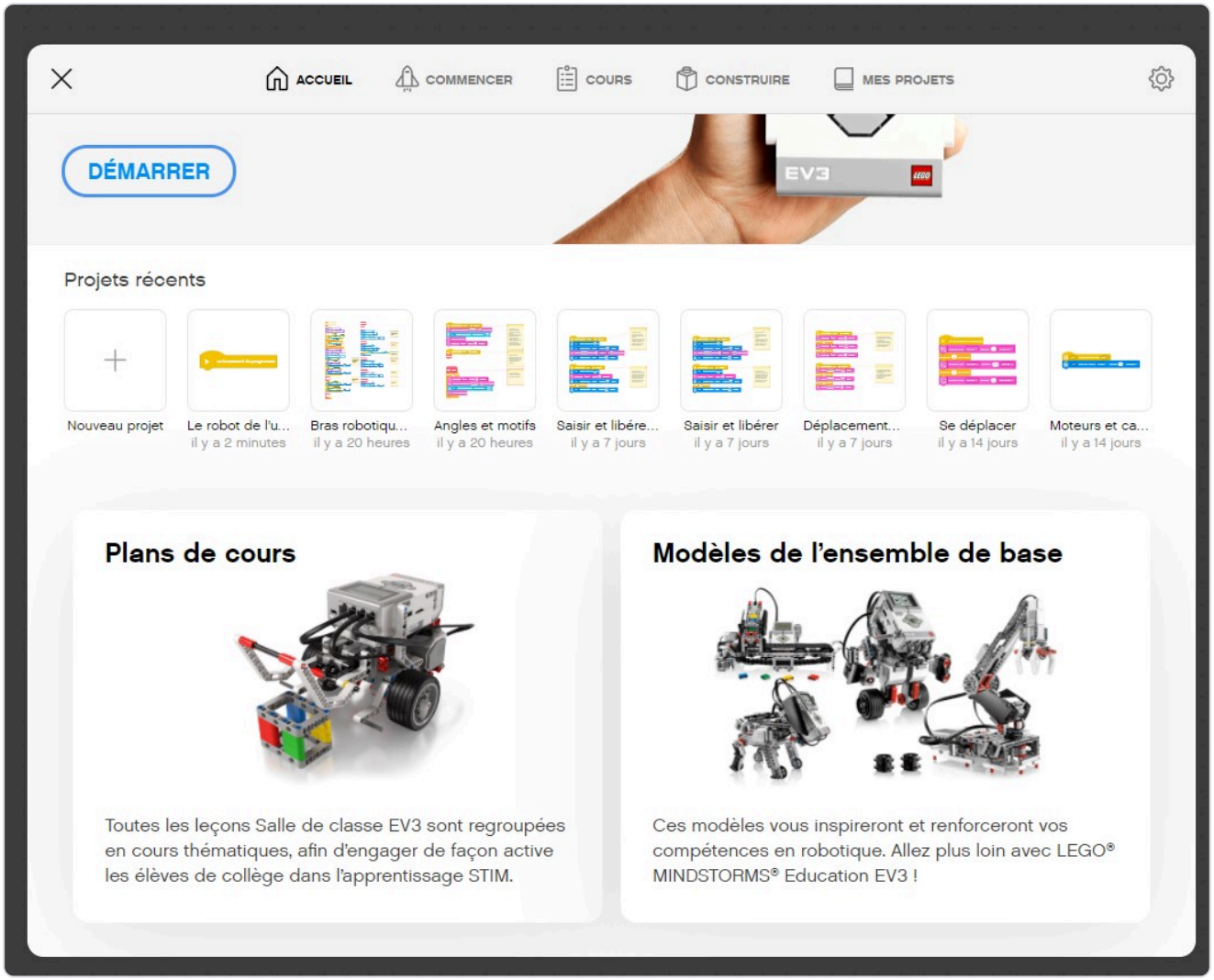
Laboratoire d'ingénierie (dans COURS)

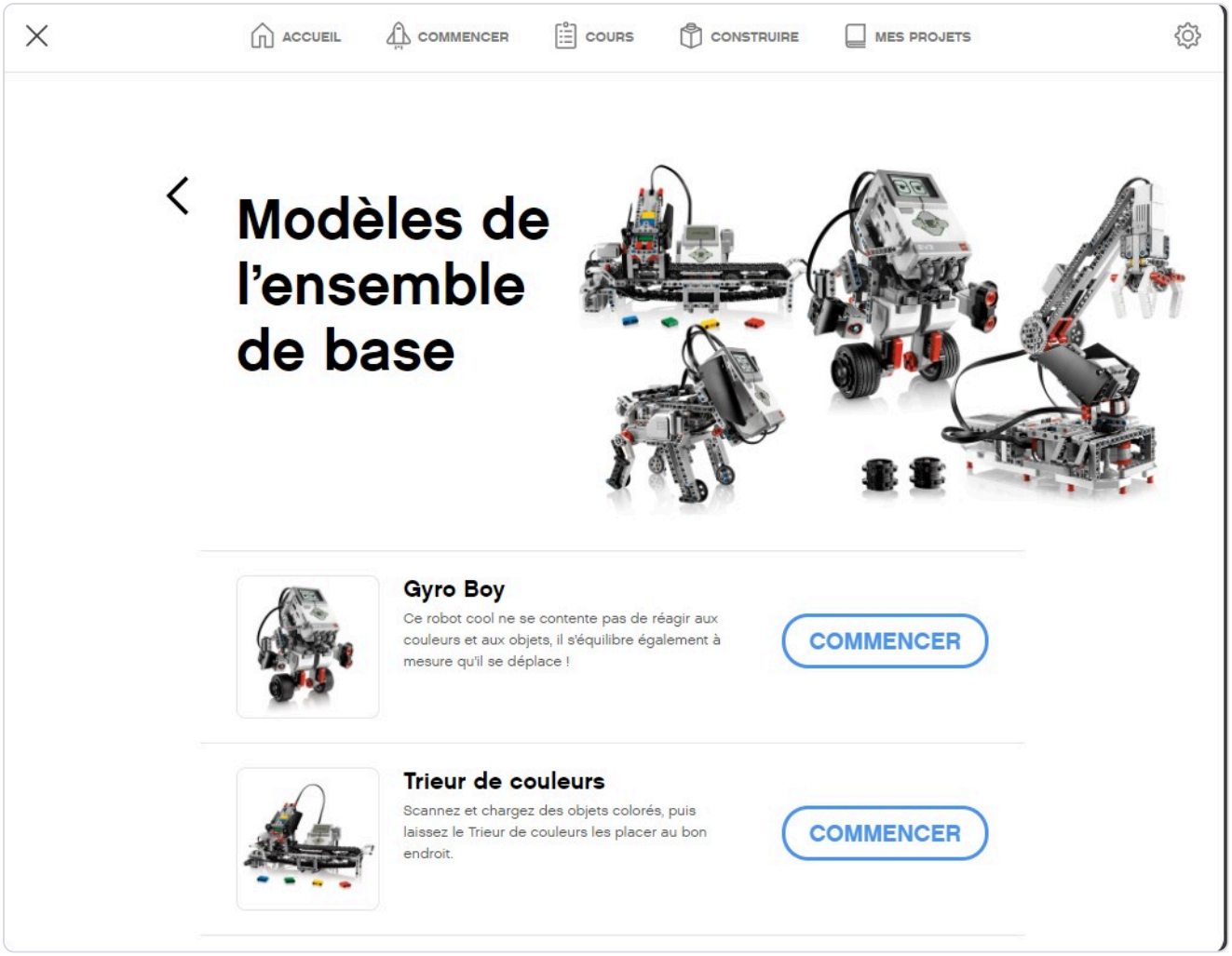
- Méthode expérimentale
- Étude de la force et du mouvement
- Projets de type scientifique



Modèles de l'ensemble de base (dans CONSTRUIRE)

- **Gyro Boy** : robot qui s'équilibre tout seul !
- **Trieur de couleurs** : robot qui classe des objets par couleur
- Et d'autres constructions créatives...


ACCUEIL
COMMENCER
COURS
CONSTRUIRE
MES PROJETS
DÉMARRER
EV3
Projets récents
Nouveau projet
Le robot de l'u...
il y a 2 minutes
Bras robotiqu...
il y a 20 heures
Angles et motifs
il y a 20 heures
Saisir et libére...
il y a 7 jours
Saisir et libérer
il y a 7 jours
Déplacement...
il y a 7 jours
Se déplacer
il y a 14 jours
Moteurs et ca...
il y a 14 jours
Plans de cours
Toutes les leçons Salle de classe EV3 sont regroupées en cours thématiques, afin d'engager de façon active les élèves de collège dans l'apprentissage STIM.
Modèles de l'ensemble de base
Ces modèles vous inspireront et renforceront vos compétences en robotique. Allez plus loin avec LEGO® MINDSTORMS® Education EV3 !



Avec notre matériel

On dispose aussi de :

- **Chenilles** → transformer le robot en char
- **Roues supplémentaires** → robot 4 roues motrices
- **Pièces pour bras robotique** → construire une pince articulée

Ressources communautaires

La communauté LEGO Mindstorm est très active ! Quelques pistes :

Sites avec plans de robots :

- LEGO Education — Plans officiels
- EV3Lessons.com — Tutoriels et défis
- Robot Square — Designs populaires
- Builderdude35 — Chaîne YouTube avec tutoriels

Mots-clés pour chercher :

- "EV3 robot design"
- "Mindstorm EV3 building instructions"
- "EV3 challenge tutorial"

💡 *YouTube regorge de vidéos de constructions incroyables. Inspire-toi !*

CHAPITRE 3

Introduction à Processing

> 🎯 ***Objectif*** *: Comprendre ce qu'est Processing, pourquoi on l'utilise dans ce cours, l'installer et découvrir l'interface. À la fin de cette introduction, tu seras capable de lancer ton premier programme.*

Slides du chapitre

- Processing.pdf

Le chaînon manquant

Dans ce cours, tu as déjà programmé avec **Scratch** : tu as assemblé des blocs colorés pour créer des programmes. C'est super pour comprendre la logique, mais dans le monde réel, les programmes s'écrivent avec du **texte**.

Le problème ? Passer directement de Scratch à un langage comme C++ ou Python, c'est un peu comme passer du vélo à la moto sur l'autoroute. Il faut une étape intermédiaire.

Processing, c'est exactement ça : un langage de programmation **textuel** conçu pour être accessible aux débutants, avec des résultats **visuels immédiats**. Tu écris du code, tu cliques sur Play, et tu vois directement le résultat à l'écran.

Un peu d'histoire

Processing est né en **2001** au **MIT** (Massachusetts Institute of Technology), créé par **Ben Fry** et **Casey Reas**. Leur objectif : permettre aux **artistes**, designers et non-programmeurs de créer des œuvres visuelles avec du code, sans avoir besoin d'un diplôme en informatique.

Le projet a tellement bien marché qu'il a reçu le **Golden Nica** du prestigieux festival Ars Electronica en 2005. Aujourd'hui, Processing est utilisé dans le monde entier pour l'art numérique, la visualisation de données, et surtout... **l'enseignement de la programmation**.

Pourquoi Processing dans un cours de Drones & Robotique ?

Voici le secret : la syntaxe de Processing est **quasiment identique** à celle du C++ utilisé pour programmer les cartes Arduino.

Processing	Arduino (C++)
`void setup() { }`	`void setup() { }`
`void draw() { }`	`void loop() { }`
`int x = 100;`	`int x = 100;`
`if (x > 50) { }`	`if (x > 50) { }`
`for (int i = 0; i < 10; i++)`	`for (int i = 0; i < 10; i++)`

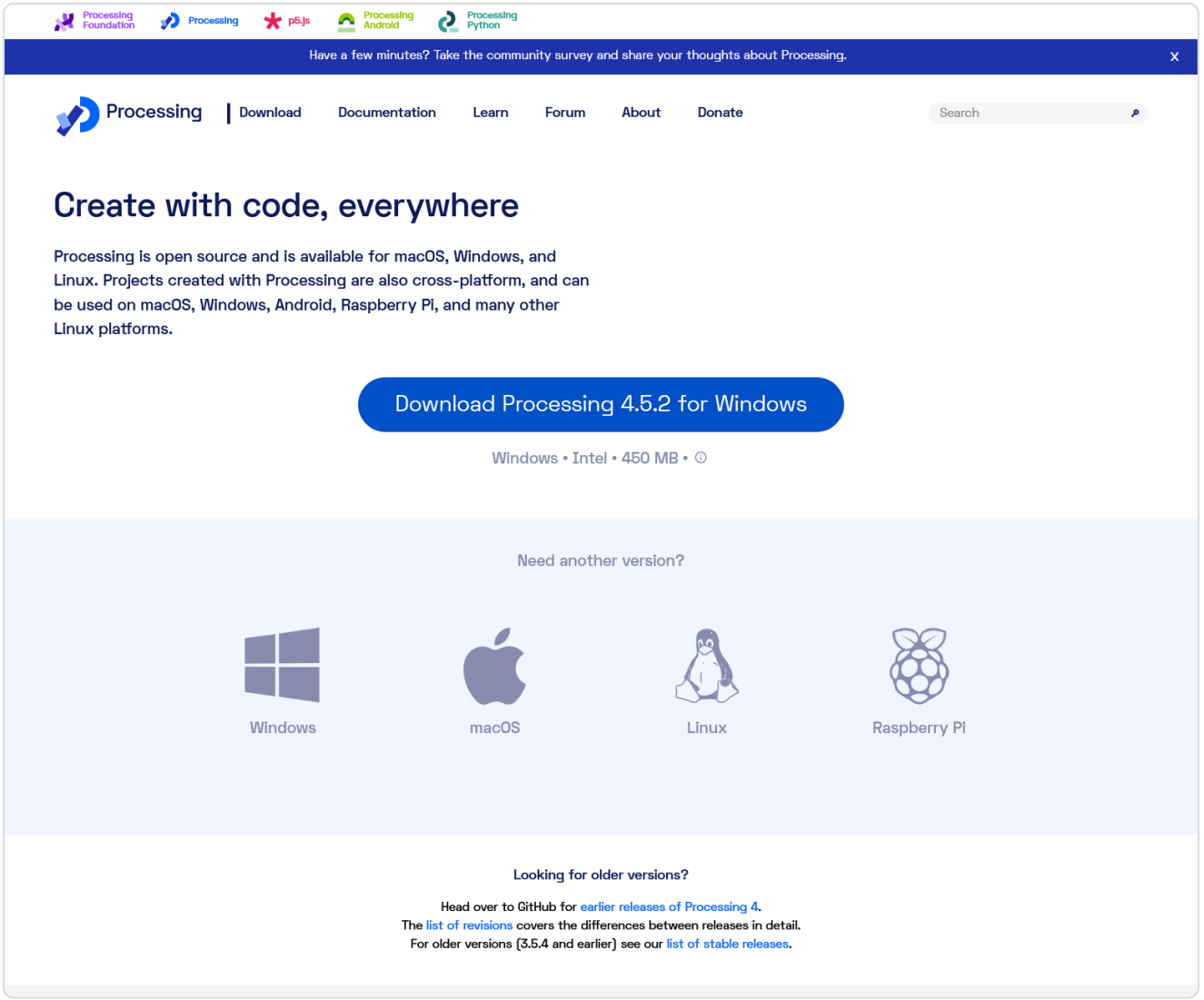
Tu vois ? C'est **le même langage** (ou presque). Techniquement, Processing est basé sur Java, mais sa syntaxe est tellement proche du C++ que ce que tu apprends ici, tu le réutiliseras **directement** quand on passera à Arduino !

💡 **Fun fact** : Processing peut même être utilisé pour programmer directement des cartes Arduino, grâce à un mode intégré.

Installer Processing

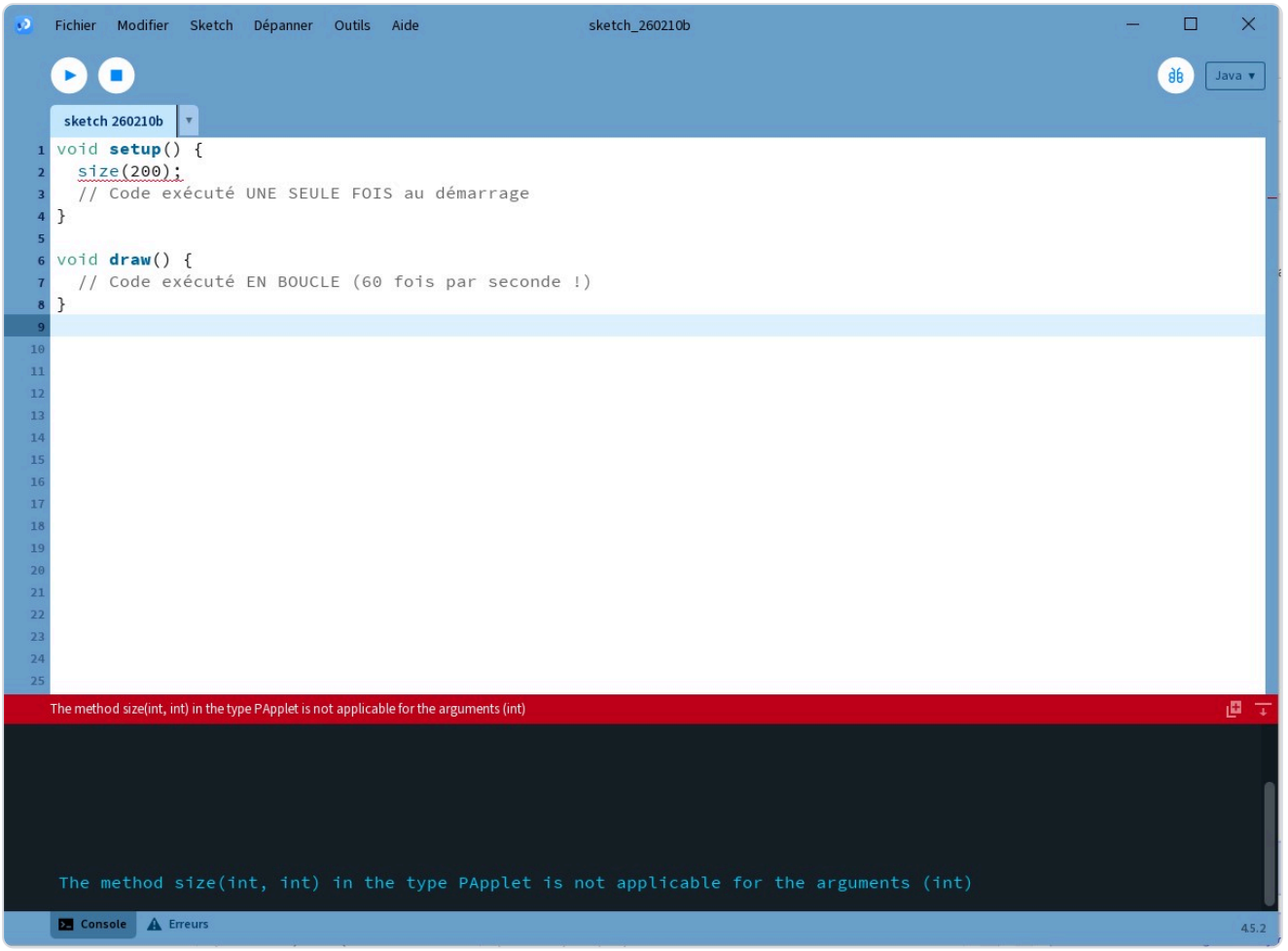
Téléchargement

1. Va sur **https://processing.org/download**
2. Clique sur le bouton de téléchargement correspondant à ton système (Windows, Mac ou Linux)
3. **Lance le fichier téléchargé** — c'est tout, pas besoin de décompresser ni de configurer quoi que ce soit !
4. Ouvre Processing



L'interface

Quand tu ouvres Processing, voici ce que tu vois :



Zone	Rôle
Zone de code (grande zone blanche)	C'est là que tu écris ton programme
Bouton ▶ Play (en haut à gauche)	Lance ton programme
Bouton ■ Stop	Arrête ton programme
Console (zone noire en bas)	Affiche les messages et les erreurs
Fenêtre de résultat	S'ouvre quand tu cliques sur Play — c'est ton canevas !

💡 **Astuce** : Si tu vois du texte rouge dans la console, c'est une **erreur**. Pas de panique ! Lis le message, il te dit souvent quelle ligne pose problème.

La structure d'un programme Processing

Tout programme Processing a la même structure de base :

```
void setup() {
  // Code exécuté UNE SEULE FOIS au démarrage
}

void draw() {
  // Code exécuté EN BOUCLE (60 fois par seconde !)
}
```

Le parallèle avec Scratch

Scratch	Processing	Rôle
🚩 Quand le drapeau vert est cliqué	`void setup() { }`	S'exécute une seule fois au début
🔁 Répéter indéfiniment	`void draw() { }`	S'exécute en boucle, encore et encore
La scène (arrière-plan)	Le **canevas** (canvas)	La zone où on dessine

Pour créer un canevas, on utilise `size()` dans le `setup()` :

```
void setup() {
  size(400, 300);  // Crée un canevas de 400 pixels de large et 300 de haut
}

void draw() {
  // On dessinera ici
}
```

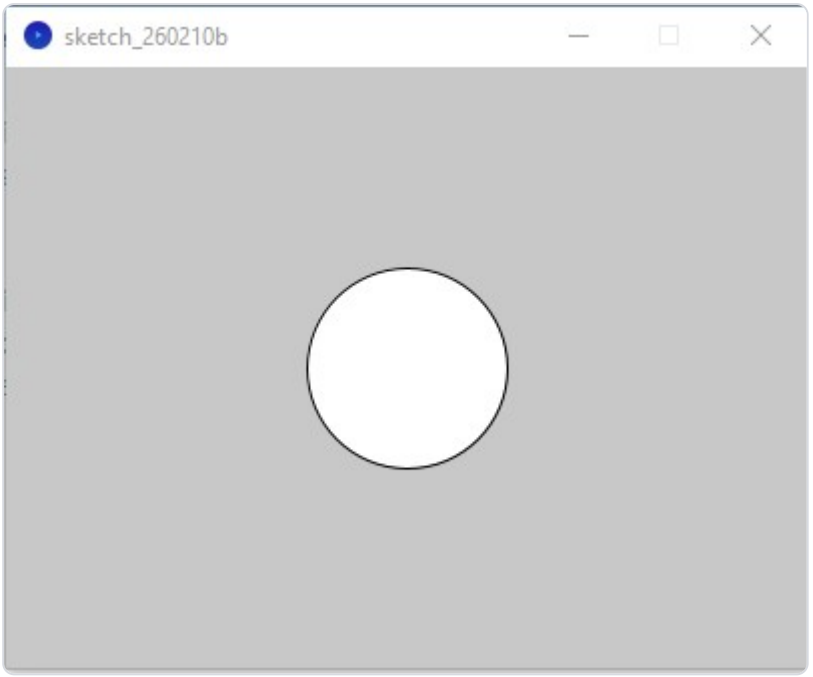
⚠️ **Attention** : `size()` doit toujours être la **première instruction** dans `setup()`.

Ton premier programme

Tape ce code dans Processing et clique sur ▶ **Play** :

```
void setup() {
  size(400, 300);
}

void draw() {
  background(200);
  ellipse(200, 150, 100, 100);
}
```



Tu devrais voir une fenêtre grise avec un cercle blanc au centre. Félicitations, tu viens d'écrire ton premier programme textuel ! 🎉

Ne t'inquiète pas si tu ne comprends pas encore chaque ligne — on va tout décortiquer dans les prochains chapitres.

Ce qu'on va apprendre dans les prochains chapitres

Voici un aperçu de la suite. Chaque concept a son équivalent en Scratch que tu connais déjà :

Chapitre	Concept	En Scratch	En Processing
0b	**Outils de dessin**	Stylo (couleur, taille)	`rect()`, `ellipse()`, `stroke()`, `fill()`
0c	**Couleurs**	Couleur du stylo / du lutin	`fill(255, 0, 0)`, `background()`
1	**Premières formes**	Dessiner avec le stylo	Combiner formes et couleurs
2	**Variables**	Bloc "mettre maVariable à 10"	`int x = 10;`
3	**Conditions**	Bloc "Si ... Alors ... Sinon"	`if (x > 5) { } else { }`
4	**Boucles**	Bloc "Répéter 10 fois"	`for (int i = 0; i < 10; i++)`
5	**Fonctions**	Bloc "Mes blocs"	`void dessinerMaison(int x, int y)`
6	**Animation**	Mouvement continu	Variable qui change dans `draw()`

Les types de données (un avant-goût)

En Scratch, une variable peut contenir n'importe quoi. En Processing (comme en C++ et Arduino), chaque variable a un **type** qui dit quel genre de valeur elle contient :

Type	Ce que ça stocke	Exemple
`int`	Un nombre **entier**	`int age = 25;`
`float`	Un nombre **à virgule**	`float prix = 9.99;`
`boolean`	**Vrai** ou **Faux**	`boolean estVisible = true;`
`String`	Du **texte**	`String nom = "Alice";`
`color`	Une **couleur**	`color rouge = color(255, 0, 0);`

Il en existe d'autres qu'on découvrira au fur et à mesure (`char`, `long`, `double`...). L'important pour l'instant : quand tu crées une variable, tu dois toujours dire son type en premier.

```
int x = 100;        // ✅ Correct
float y = 3.14;     // ✅ Correct
x = "bonjour";      // ❌ Erreur ! x est un int, pas du texte
```

On verra tout ça en détail dans le chapitre sur les variables.

Pour aller plus loin

Si tu veux explorer Processing par toi-même :

- **Documentation officielle** : https://processing.org/reference — la liste complète de toutes les fonctions
- **The Coding Train** (YouTube) : Daniel Shiffman explique Processing de façon fun et accessible
- **Nature of Code** (https://natureofcode.com) : un livre gratuit pour les plus avancés, mêlant code et simulation physique
- **OpenProcessing** (https://openprocessing.org) : des milliers de créations pour s'inspirer

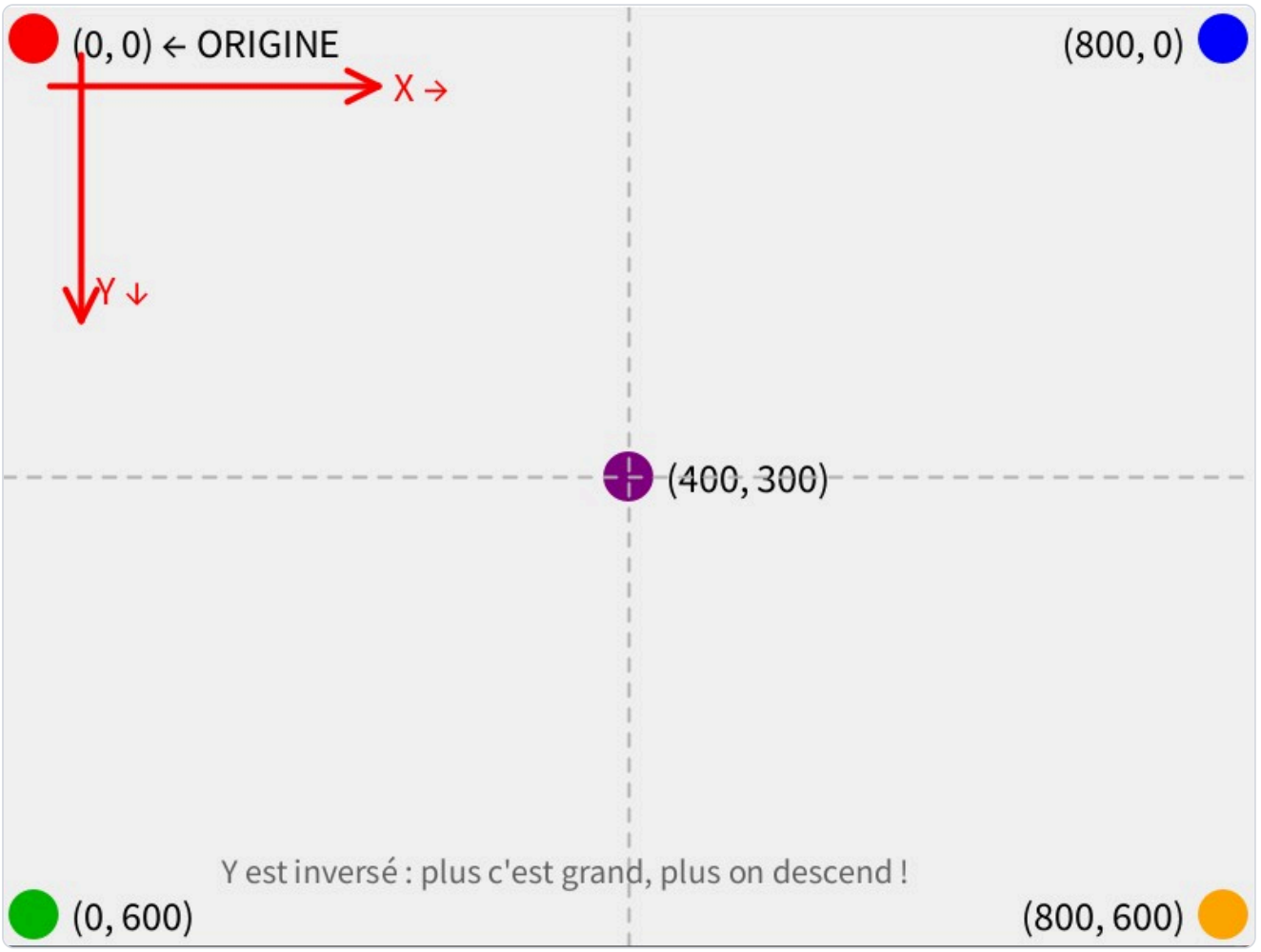
3.1 Les Outils de Dessin

> 🎯 ***Objectif*** *: Maîtriser le système de coordonnées, dessiner toutes les formes de base, et comprendre le fonctionnement du remplissage et du contour. À la fin de ce chapitre, tu seras capable de dessiner n'importe quelle combinaison de formes sur ton canevas.*

Le système de coordonnées

L'origine est en haut à gauche

C'est différent des maths ! En Processing, le point **(0, 0)** est en **haut à gauche** du canevas. L'axe X va vers la **droite** et l'axe Y va vers le **bas**.



Quand on écrit `ellipse(100, 200, 50, 50)`, on dessine un cercle dont le centre est à **100 pixels depuis la gauche** et **200 pixels depuis le haut**.

⚠️ **Piège classique** : l'axe Y est **inversé** par rapport aux cours de maths. Plus la valeur de Y est grande, plus on **descend** !

Direction	Valeur qui augmente
→ Vers la droite	X augmente
↓ Vers le bas	Y augmente
← Vers la gauche	X diminue
↑ Vers le haut	Y diminue

Les formes de base

Processing met à ta disposition plusieurs fonctions pour dessiner. Voici les principales :

Le point

```
point(x, y);
```

Dessine un seul pixel à la position (x, y). Pas très visible ! Utilise `strokeWeight()` pour le rendre plus gros.

```
strokeWeight(8);
point(200, 150);   // Un gros point au centre
```

La ligne

```
line(x1, y1, x2, y2);
```

Trace une ligne entre le point (x1, y1) et le point (x2, y2).

```
line(50, 50, 350, 250);  // Ligne diagonale
```

Le rectangle

```
rect(x, y, largeur, hauteur);
```

Dessine un rectangle. Par défaut, **(x, y) est le coin supérieur gauche**.

```
rect(100, 80, 200, 120);  // Rectangle à partir du coin (100, 80)
```

💡 Pour que (x, y) soit le **centre** du rectangle au lieu du coin, utilise :

```
rectMode(CENTER);
rect(200, 150, 200, 120);  // (200, 150) est maintenant le CENTRE
```

L'ellipse (et le cercle)

```
ellipse(x, y, largeur, hauteur);
```

Dessine une ellipse **centrée** sur (x, y). Si largeur = hauteur, c'est un **cercle**.

```
ellipse(200, 150, 100, 100);  // Cercle de diamètre 100
ellipse(200, 150, 150, 80);   // Ellipse aplatie
```

Le triangle

```
triangle(x1, y1, x2, y2, x3, y3);
```

Dessine un triangle en reliant trois points.

```
triangle(200, 50, 100, 250, 300, 250);  // Triangle pointant vers le haut
```

L'arc (portion de cercle)

```
arc(x, y, largeur, hauteur, angleDebut, angleFin);
```

Dessine un arc de cercle. Les angles sont en **radians**, pas en degrés.

Angle	Radians	Constante Processing
0°	0	`0`
90°	π/2	`HALF_PI`
180°	π	`PI`
270°	3π/2	`PI + HALF_PI`
360°	2π	`TWO_PI`

```
arc(200, 150, 160, 160, 0, PI);  // Demi-cercle (de 0° à 180°)
```

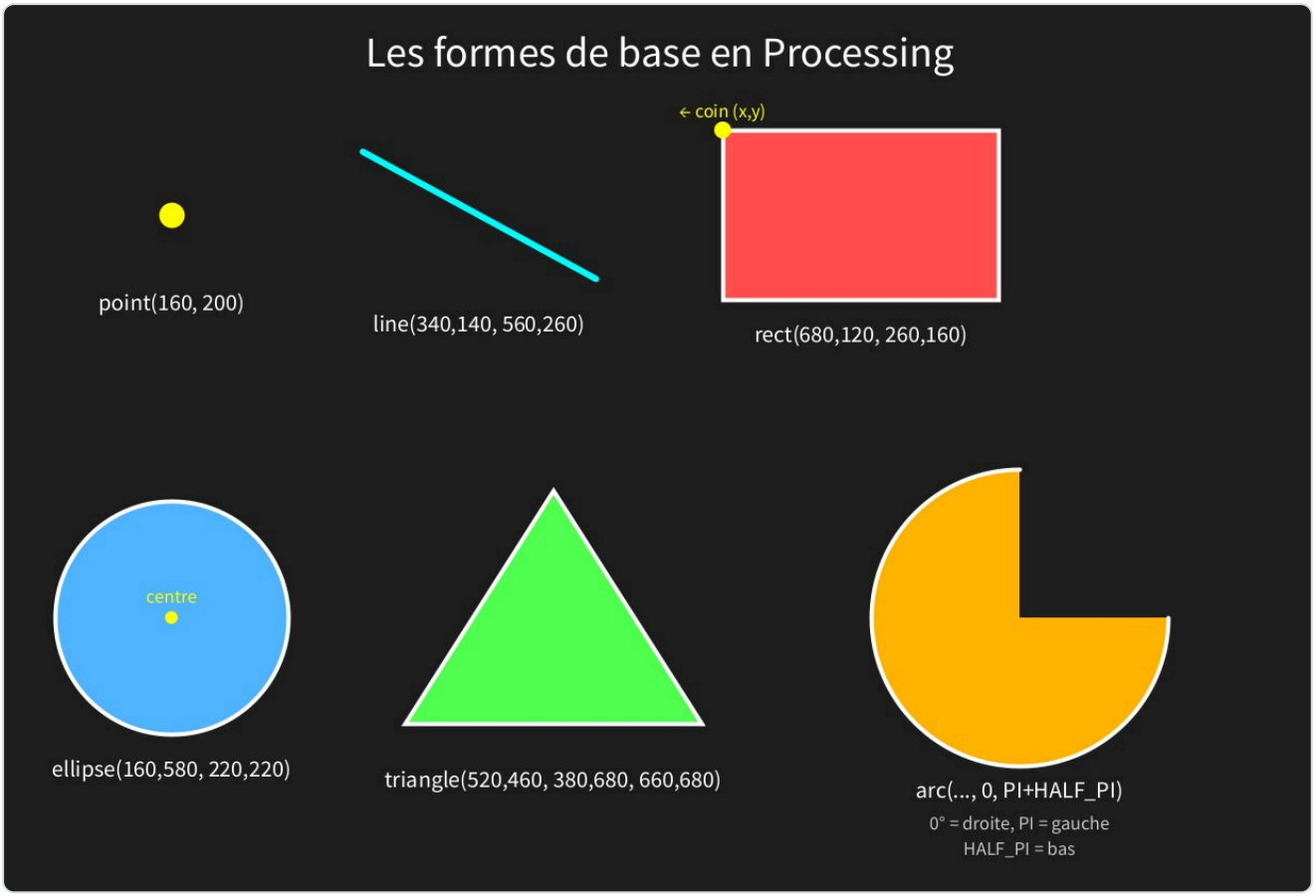


Tableau récapitulatif

Fonction	Paramètres	Dessine...
`point(x, y)`	Position	Un point
`line(x1, y1, x2, y2)`	Deux points	Une ligne
`rect(x, y, l, h)`	Coin + dimensions	Un rectangle
`ellipse(x, y, l, h)`	Centre + dimensions	Une ellipse/cercle
`triangle(x1,y1, x2,y2, x3,y3)`	Trois points	Un triangle
`arc(x, y, l, h, début, fin)`	Centre + dimensions + angles	Un arc
`background(couleur)`	Couleur	Remplit tout le fond

Le remplissage et le contour

Processing fonctionne comme si tu avais deux outils dans la main :

🎨 Le pot de peinture : `fill()`

`fill()` définit la couleur de **remplissage** (l'intérieur) de toutes les formes dessinées après.

```
fill(200);              // Remplissage gris clair
rect(50, 50, 100, 80);  // Ce rectangle sera gris clair à l'intérieur
```

Pour désactiver le remplissage (forme "vide", transparente) :

```
noFill();
```

✏️ Le stylo : `stroke()` et `strokeWeight()`

`stroke()` définit la couleur du **contour** (le trait autour des formes).

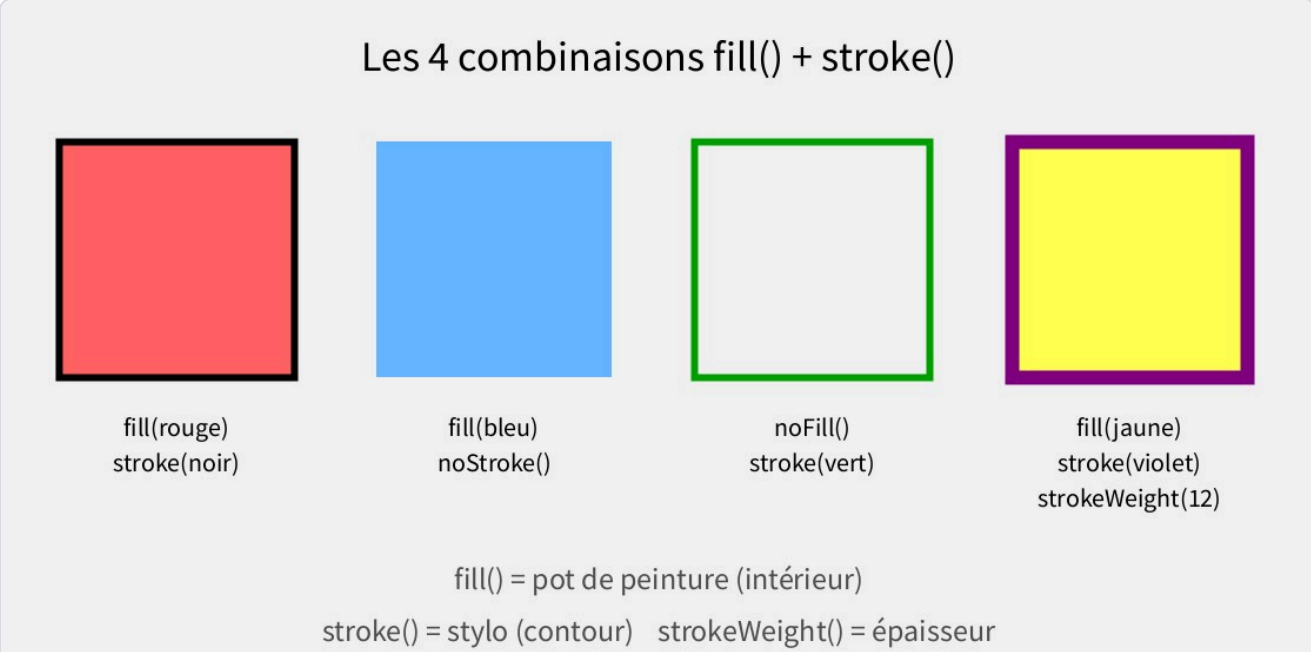
`strokeWeight()` définit l'**épaisseur** du trait en pixels.

```
stroke(0);              // Contour noir
strokeWeight(4);         // Trait de 4 pixels d'épaisseur
rect(50, 50, 100, 80);  // Ce rectangle aura un gros contour noir
```

Pour désactiver le contour :

```
noStroke();
```

Les 4 combinaisons possibles



Combinaison	Code	Résultat
Remplissage + contour	`fill(200); stroke(0);`	Forme grise avec bord noir
Remplissage seul	`fill(200); noStroke();`	Forme grise, pas de bord
Contour seul	`noFill(); stroke(0);`	Juste le contour, intérieur vide
Les deux personnalisés	`fill(200); stroke(0); strokeWeight(5);`	Forme grise avec gros bord noir

💡 Règle importante : l'effet "feutre"

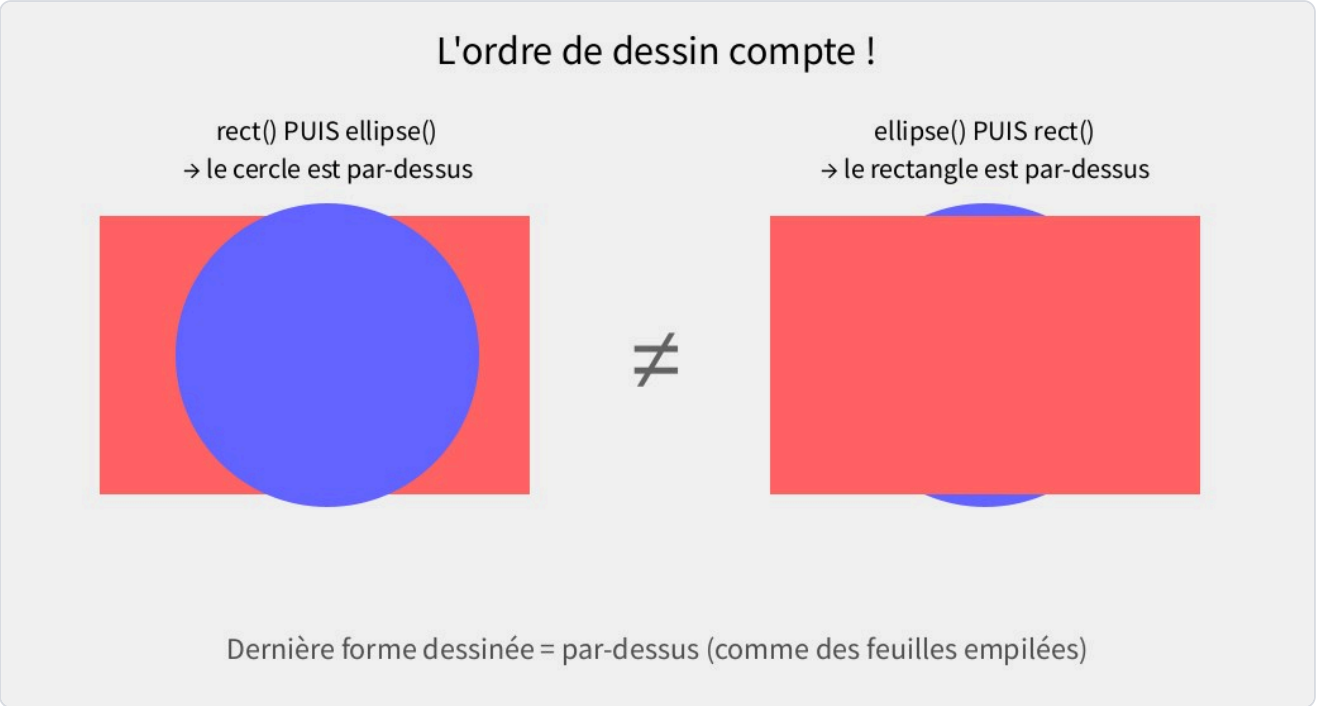
`fill()` et `stroke()` s'appliquent à **tout ce qui est dessiné après**, jusqu'au prochain appel. C'est comme si tu changeais de feutre : une fois que tu prends le rouge, tout ce que tu dessines est rouge jusqu'à ce que tu changes.

```
fill(200);                  // "Je prends le feutre gris"
rect(20, 20, 80, 80);       // Rectangle gris
ellipse(200, 60, 80, 80);   // Cercle gris AUSSI !

fill(100);                  // "Je change pour le feutre gris foncé"
rect(300, 20, 80, 80);      // Rectangle gris foncé
```

L'ordre des instructions compte !

En Processing, les formes sont dessinées **les unes par-dessus les autres**, dans l'ordre du code. La dernière forme dessinée est **par-dessus** toutes les autres.




```
// Le cercle sera PAR-DESSUS le rectangle
fill(200);
rect(50, 50, 200, 100);     // Dessiné en premier → en dessous

fill(100);
ellipse(150, 100, 120, 120); // Dessiné en dernier → par-dessus
```

C'est comme quand tu empiles des feuilles de papier : la dernière posée est celle qu'on voit.

⚠️ **Conséquence** : si tu veux qu'un fond soit derrière tout, dessine-le **en premier** ! C'est le rôle de `background()` qu'on met au début du `draw()`.

Résumé

Outil	Fonction	Ce que ça fait
🎨 Remplissage	`fill(couleur)`	Couleur de l'intérieur des formes
🎨 Pas de remplissage	`noFill()`	Formes transparentes
✏️ Contour	`stroke(couleur)`	Couleur du trait/bord
✏️ Épaisseur	`strokeWeight(pixels)`	Épaisseur du trait

Outil	Fonction	Ce que ça fait
✏️ Pas de contour	`noStroke()`	Pas de trait autour des formes
📐 Mode rectangle	`rectMode(CENTER)`	Change le point de référence du rectangle
🖼️ Fond	`background(couleur)`	Remplit tout le canevas

💡 **Astuce finale** : quand tu te perds, rappelle-toi que chaque forme a besoin de **trois choses** :

1. Un **remplissage** (`fill()` ou `noFill()`)
2. Un **contour** (`stroke()` / `strokeWeight()` ou `noStroke()`)
3. Sa **position et taille** (les paramètres de la fonction)

3.2 Les Couleurs

> 🎯 ***Objectif*** *: Maîtriser le système de couleurs de Processing, des niveaux de gris jusqu'aux couleurs RGB avec transparence. À la fin de ce chapitre, tu seras capable de colorier n'importe quelle forme avec la couleur exacte que tu veux.*

Les niveaux de gris

La façon la plus simple d'utiliser les couleurs en Processing : **une seule valeur** entre 0 et 255.

Valeur	Couleur
`0`	⬛ Noir complet
`64`	Gris foncé
`128`	Gris moyen
`192`	Gris clair
`255`	⬜ Blanc complet

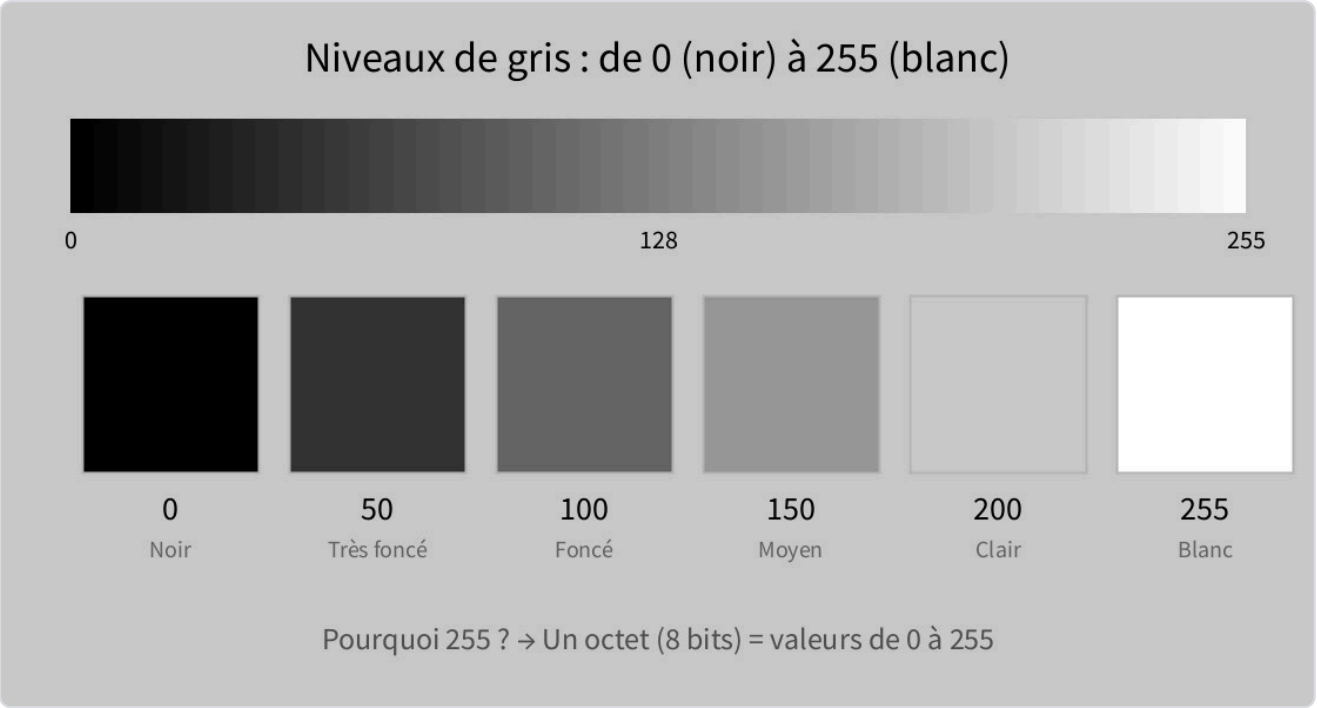
```
background(240);        // Fond gris très clair

fill(0);                // Noir
rect(20, 20, 80, 80);

fill(128);              // Gris moyen
rect(120, 20, 80, 80);

fill(255);              // Blanc
rect(220, 20, 80, 80);
```

💡 **Pourquoi 255 et pas 100 ?** Parce qu'un octet (8 bits) peut stocker des valeurs de 0 à 255. C'est un standard en informatique que tu retrouveras partout : écrans, images, web...



Les couleurs RGB

Pour obtenir de vraies couleurs, on utilise le système **RGB** : **R**ouge, **V**ert (Green), **B**leu. Chaque composante va de 0 à 255.

```
fill(rouge, vert, bleu);
```

Le principe est celui de la **synthèse additive** : on mélange de la lumière, pas de la peinture ! C'est comme les pixels de ton écran qui combinent trois petites LEDs (rouge, verte, bleue).

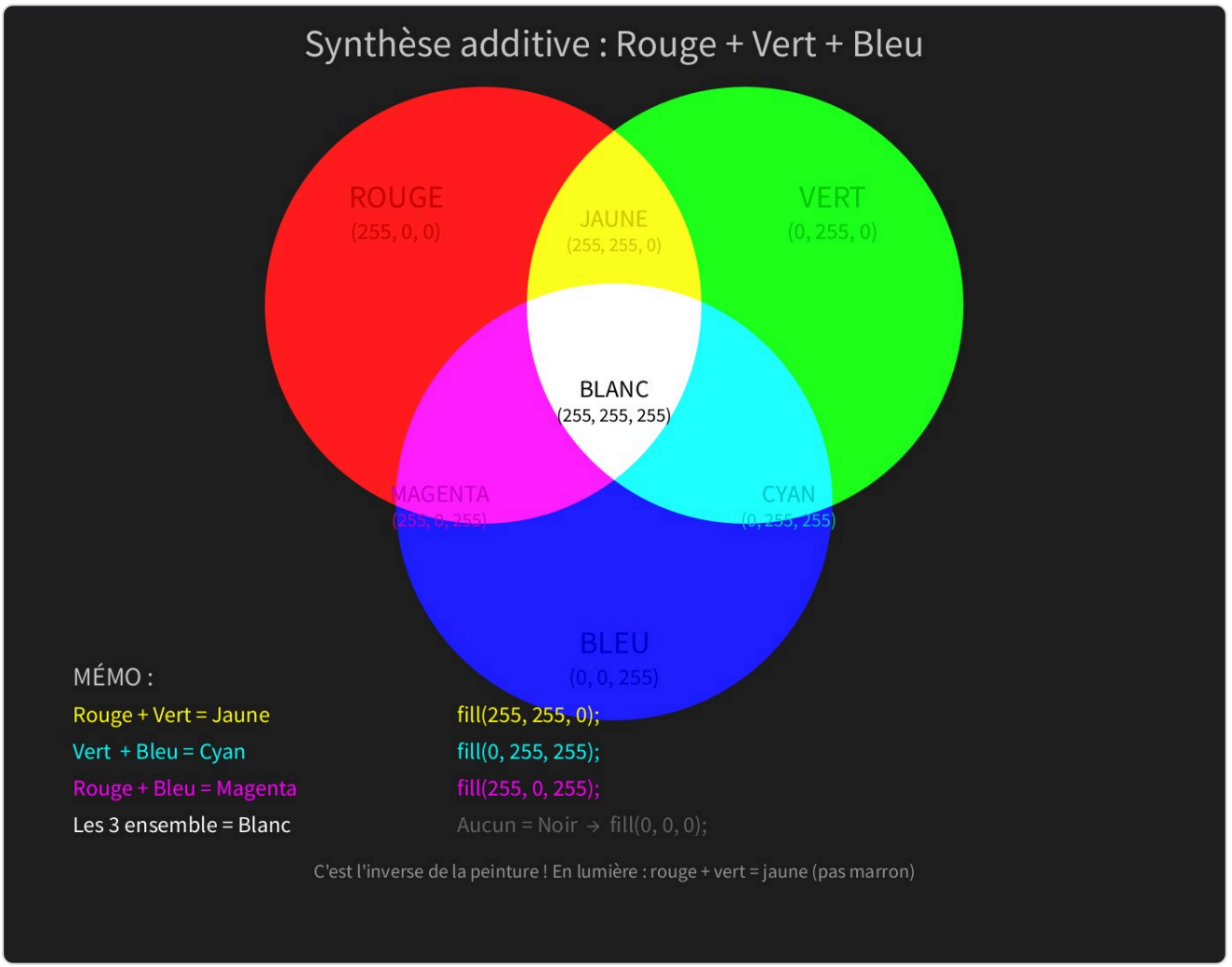
Les couleurs primaires

```
fill(255, 0, 0);    // 🟥 Rouge pur — que du rouge, pas de vert, pas de bleu
fill(0, 255, 0);    // 🟩 Vert pur
fill(0, 0, 255);    // 🟦 Bleu pur
```

Les mélanges

```
fill(255, 255, 0);   // 🟨 Rouge + Vert = Jaune
fill(0, 255, 255);   //      Vert + Bleu = Cyan
fill(255, 0, 255);   // 🟪 Rouge + Bleu = Magenta/Violet
fill(255, 255, 255); // ⬜ Les trois ensemble = Blanc
fill(0, 0, 0);       // ⬛ Aucun = Noir
```

⚠️ **Attention** : mélanger de la lumière, c'est l'inverse de la peinture ! En peinture, rouge + vert = marron. En lumière (RGB), rouge + vert = **jaune**. C'est surprenant mais c'est comme ça que fonctionnent tous les écrans.



Mémo des couleurs courantes

Voici les codes RGB des couleurs que tu utiliseras le plus souvent :

Couleur	Code RGB	Astuce pour retenir
⬛ Noir	`0, 0, 0`	Tout à zéro

Couleur	Code RGB	Astuce pour retenir
⬜ Blanc	`255, 255, 255`	Tout au max
🟥 Rouge	`255, 0, 0`	Premier à fond
🟩 Vert	`0, 255, 0`	Deuxième à fond
🟦 Bleu	`0, 0, 255`	Troisième à fond
🟨 Jaune	`255, 255, 0`	Rouge + Vert
🟧 Orange	`255, 165, 0`	Rouge + un peu de Vert
🟪 Violet	`128, 0, 128`	Rouge + Bleu (à moitié)
Cyan	`0, 255, 255`	Vert + Bleu
🦩 Rose	`255, 192, 203`	Beaucoup de tout, surtout Rouge
🤎 Brun	`139, 69, 19`	Rouge dominant, peu de Vert
🪨 Gris	`128, 128, 128`	Les trois égaux (entre noir et blanc)

💡 **Astuce** : pas besoin de retenir tout ça par cœur ! Processing a un outil intégré pour choisir les couleurs. Va dans `Outils > Sélecteur de couleurs` et clique sur la couleur que tu veux — il te donne le code RGB.

Où utiliser les couleurs ?

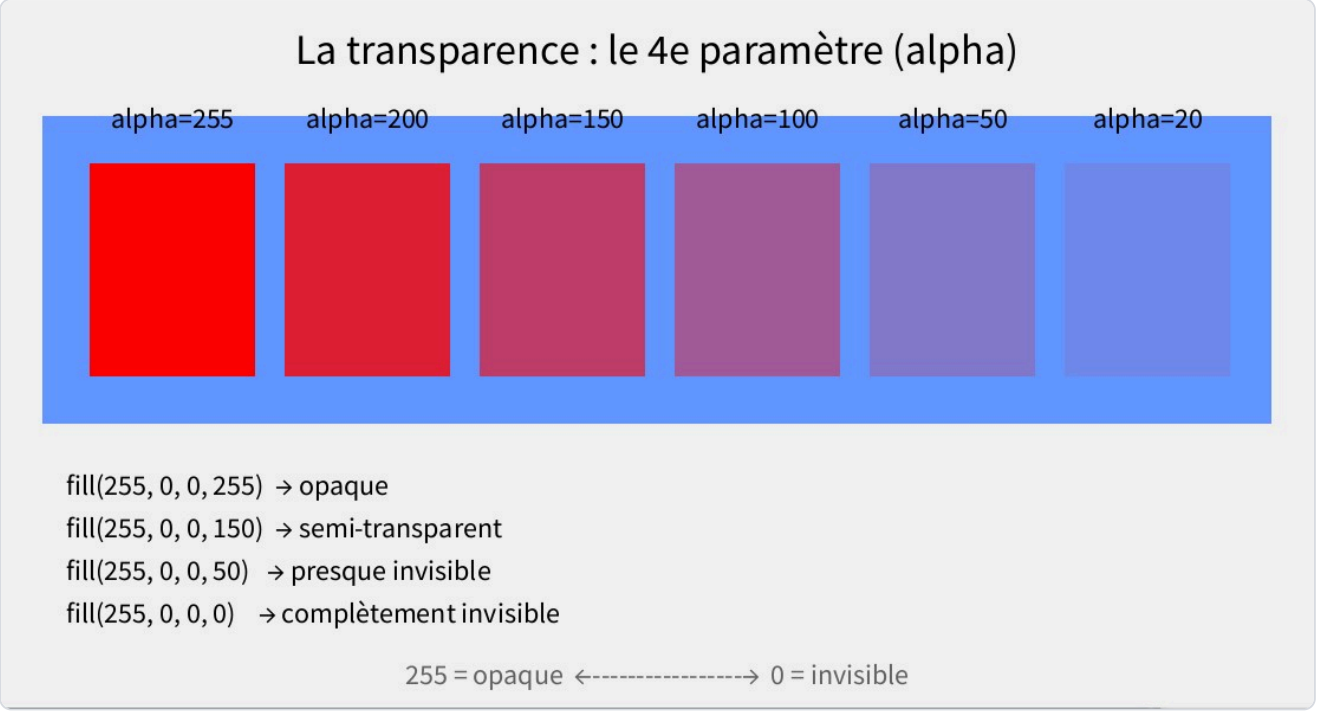
Les couleurs s'appliquent à trois endroits :

Fonction	Ce qu'elle colorie	Exemple
`background(r, g, b)`	Le **fond** du canevas	`background(135, 206, 235);` → ciel bleu
`fill(r, g, b)`	L'**intérieur** des formes	`fill(255, 0, 0);` → formes rouges
`stroke(r, g, b)`	Le **contour** des formes	`stroke(0, 0, 255);` → contour bleu

Chacune de ces fonctions accepte les mêmes formats :

```
fill(128);            // 1 valeur  → niveau de gris
fill(255, 0, 0);      // 3 valeurs → couleur RGB
fill(255, 0, 0, 128); // 4 valeurs → couleur RGB + transparence
```

La transparence



On peut ajouter un **4e nombre** pour la transparence (appelée **alpha**), de 0 (invisible) à 255 (opaque) :

```
fill(255, 0, 0, 255);  // Rouge complètement opaque (par défaut)
fill(255, 0, 0, 180);  // Rouge un peu transparent
fill(255, 0, 0, 100);  // Rouge assez transparent
fill(255, 0, 0, 30);   // Rouge presque invisible
```

C'est très utile pour superposer des formes et créer des effets de profondeur ou de verre :

```
background(240);

// Cercle bleu
fill(0, 0, 255);
ellipse(180, 150, 150, 150);

// Cercle rouge semi-transparent par-dessus
fill(255, 0, 0, 120);
ellipse(250, 150, 150, 150);
```

💡 **Astuce** : la transparence fonctionne aussi avec `stroke()` :

```
stroke(0, 0, 0, 100);  // Contour noir semi-transparent
```

Le type `color`

Processing possède un type spécial pour stocker des couleurs dans des variables. C'est pratique quand tu utilises souvent la même couleur :

```
color rouge = color(255, 0, 0);
color bleuCiel = color(135, 206, 235);

background(bleuCiel);
fill(rouge);
rect(50, 50, 100, 80);
```

On verra les variables en détail dans un prochain chapitre, mais sache que ça existe !

Mini-exemple : une scène colorée



Voici un exemple qui combine tout ce qu'on a vu dans les trois chapitres d'introduction — formes, couleurs, remplissage, contour et ordre de dessin :

```
void setup() {
  size(500, 400);
}

void draw() {
  // Ciel
  background(135, 206, 235);

  // Soleil (jaune, sans contour)
  noStroke();
  fill(255, 220, 50);
  ellipse(420, 60, 80, 80);

  // Sol (vert)
  fill(80, 180, 80);
  rect(0, 300, 500, 100);

  // Maison : mur (marron avec contour)
  stroke(100, 60, 30);
  strokeWeight(2);
  fill(200, 120, 80);
  rect(150, 200, 200, 120);

  // Maison : toit (triangle rouge)
  fill(180, 40, 40);
  triangle(130, 200, 250, 120, 370, 200);

  // Maison : porte (marron foncé)
  fill(100, 60, 30);
  rect(220, 250, 50, 70);

  // Fenêtres (bleu clair)
  fill(180, 220, 255);
  rect(170, 230, 35, 35);
  rect(295, 230, 35, 35);
}
```

Remarque l'**ordre** : le ciel en premier (tout derrière), puis le sol, puis la maison, puis les détails par-dessus. Chaque élément recouvre les précédents.

Résumé

Concept	Syntaxe	Exemple
Niveau de gris	`fill(valeur)`	`fill(128)` → gris moyen
Couleur RGB	`fill(r, g, b)`	`fill(255, 0, 0)` → rouge
Avec transparence	`fill(r, g, b, a)`	`fill(255, 0, 0, 128)` → rouge transparent

Concept	Syntaxe	Exemple
Fond du canevas	`background(r, g, b)`	`background(135, 206, 235)` → ciel
Stocker une couleur	`color nom = color(r, g, b)`	`color c = color(255, 0, 0);`
Sélecteur intégré	`Outils > Sélecteur de couleurs`	Trouve n'importe quel code RGB

Valeur RGB	Règle
`0`	Rien de cette composante
`128`	Moitié
`255`	Maximum de cette composante
Les 3 égales	Niveau de gris (du noir au blanc)
Les 3 à 255	Blanc
Les 3 à 0	Noir

3.3 Premières Formes

> 🎯 ***Objectif*** *: Combiner formes, couleurs, remplissage et contour pour créer des compositions visuelles. À la fin de ce chapitre, tu seras capable de dessiner une scène complète en contrôlant précisément l'apparence de chaque élément.*

Rappel express

Dans les chapitres précédents, tu as découvert trois familles d'outils :

Famille	Ce que tu sais faire	Fonctions
📐 **Formes**	Dessiner des formes géométriques	`rect()`, `ellipse()`, `triangle()`, `line()`, `point()`, `arc()`
🎨 **Remplissage**	Colorier l'intérieur des formes	`fill()`, `noFill()`
✏️ **Contour**	Contrôler le trait autour des formes	`stroke()`, `noStroke()`, `strokeWeight()`

Ce chapitre, c'est la **mise en pratique**. On va combiner tout ça pour créer de vraies compositions, comme un artiste qui sort enfin ses pinceaux après avoir appris à les tenir.

Exercice 1 — Placement de formes

Consigne

Crée un canevas de **400×300** pixels avec un fond de la couleur de ton choix. Place dessus :

- Un **carré** (rappel : un carré, c'est un rectangle avec la même largeur et la même hauteur)
- Un **triangle**
- Une **ligne**

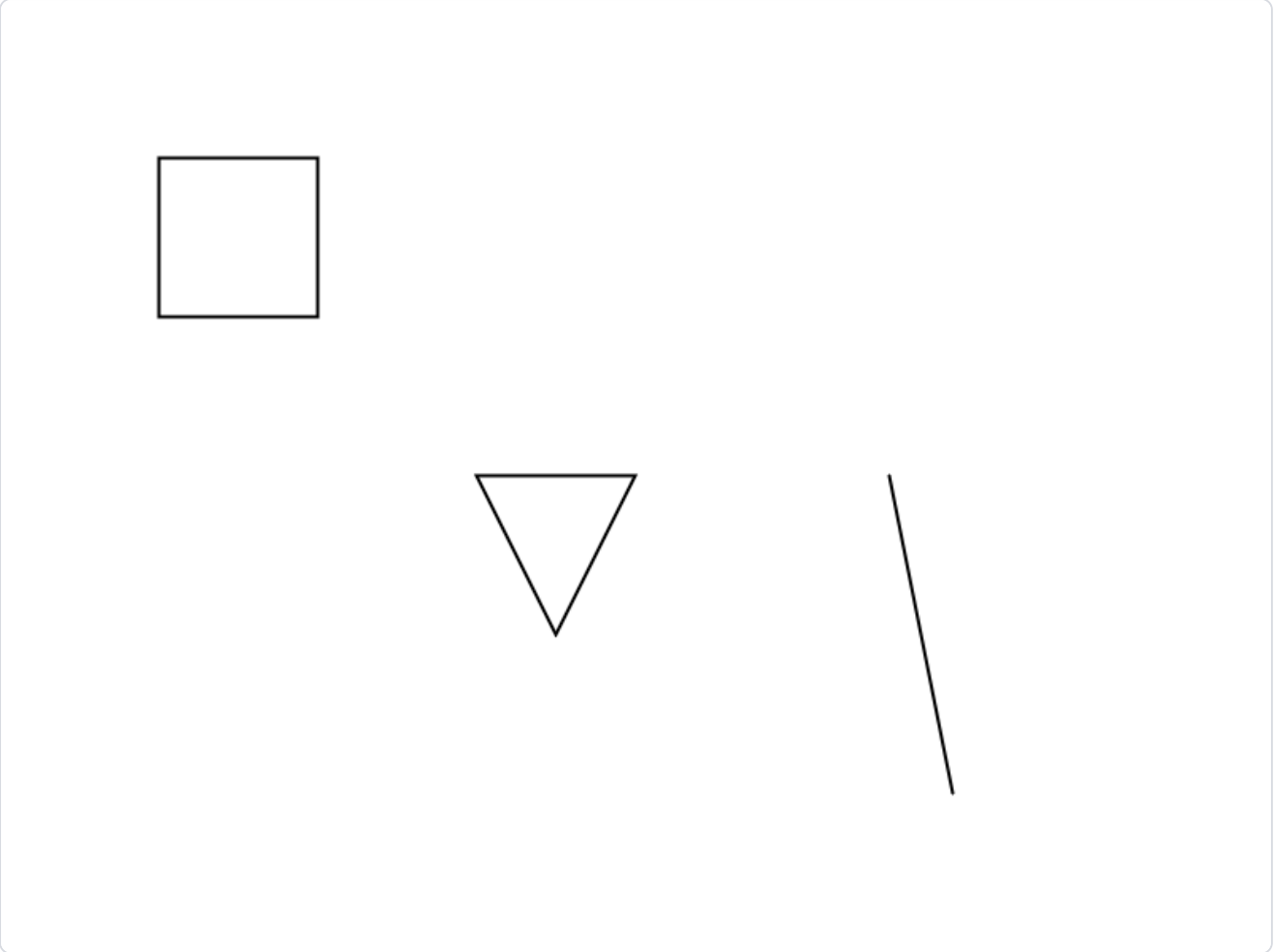
Les trois formes **ne doivent pas se chevaucher**.

Contraintes

- Utilise `size(400, 300)` dans le `setup()`
- Choisis un `background()` qui te plaît
- Laisse les couleurs par défaut (on les changera dans l'exercice 2)

Ce que tu devrais obtenir

Un canevas avec trois formes bien séparées. Par exemple :



Indices

💡 **Indice 1** — Un carré est un `rect()` dont la largeur et la hauteur sont **identiques** : `rect(50, 50, 80, 80)` fait un carré de 80×80 pixels.

💡 **Indice 2** — Ton canevas fait 400 pixels de large. Divise mentalement l'espace en trois zones : gauche (0 à ~130) pour le carré, centre (~130 à ~270) pour le triangle, droite (~270 à 400) pour la ligne.

💡 **Indice 3** — Structure du programme :

```
void setup() {
  size(400, 300);
}

void draw() {
  background(???);

  // Carré
  rect(???, ???, ???, ???);

  // Triangle
  triangle(???, ???, ???, ???, ???, ???);

  // Ligne
  line(???, ???, ???, ???);
}
```

Exercice 2 — Changement de couleurs

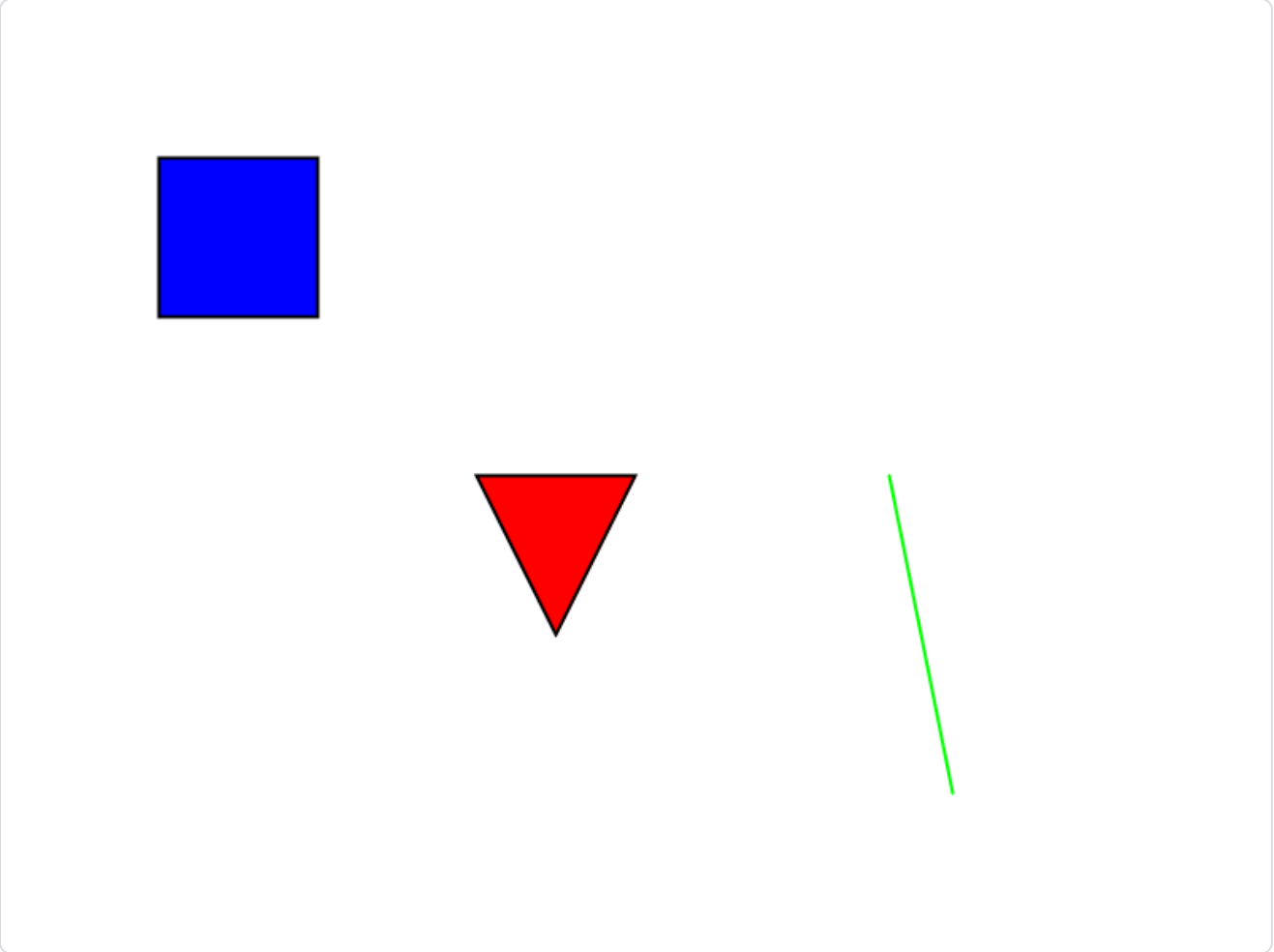
Consigne

Reprends ton code de l'exercice 1 et ajoute des couleurs :

- Le carré doit être rempli en **bleu** 🟦
- Le triangle doit être rempli en **rouge** 🟥
- La ligne doit être **verte** 🟩 et plus épaisse (au moins 3 pixels)

Contraintes

- Le carré doit avoir un **contour noir** visible
- Le triangle doit avoir un **contour noir** visible
- La ligne doit être **épaisse** (utilise `strokeWeight()`)



Indices

💡 **Indice 1** — Rappel des couleurs de base : `fill(0, 0, 255)` pour le bleu, `fill(255, 0, 0)` pour le rouge, `stroke(0, 255, 0)` pour un contour/ligne vert.

💡 **Indice 2** — Une ligne n'a **pas de remplissage** (`fill()` n'a aucun effet sur elle). Pour changer la couleur d'une ligne, c'est `stroke()` qu'il faut utiliser. Et pour l'épaissir : `strokeWeight()` .

💡 **Indice 3** — Si tu veux que le carré et le triangle aient un contour noir mais que la ligne soit verte, tu dois changer le `stroke()` entre les formes :

```
stroke(0);            // Contour noir pour le carré et le triangle
fill(0, 0, 255);
rect(...);

fill(255, 0, 0);
triangle(...);

stroke(0, 255, 0);    // On passe au vert pour la ligne
strokeWeight(4);
line(...);
```

🎯 Défi bonus — Le drapeau

Tu as terminé les exercices 1 et 2 ? Dessine le drapeau d'un pays de ton choix ! Par exemple :

- 🇧🇪 **Belgique** : trois rectangles verticaux (noir, jaune, rouge)
- 🇫🇷 **France** : trois rectangles verticaux (bleu, blanc, rouge)
- 🇯🇵 **Japon** : fond blanc + cercle rouge au centre
- 🇩🇪 **Allemagne** : trois rectangles horizontaux (noir, rouge, jaune)
- 🇮🇹 **Italie** : trois rectangles verticaux (vert, blanc, rouge)

💡 **Astuce** : pour un drapeau, utilise `noStroke()` pour éviter les traits entre les bandes.

💡 **Astuce** : pour un drapeau à 3 bandes verticales qui remplit tout le canevas, chaque bande fait un tiers de la largeur. Par exemple avec `size(300, 200)`, chaque bande fait 100 pixels.

Résumé

Ce chapitre t'a permis de mettre en pratique les trois piliers du dessin en Processing :

Pilier	Ce qu'il faut retenir
📐 **Formes**	Chaque fonction de dessin a ses propres paramètres (position, taille, points...)
🎨 **Couleurs**	`fill()` pour l'intérieur, `stroke()` pour le contour — ils restent actifs jusqu'au prochain appel
📋 **Ordre**	Ce qui est dessiné en dernier apparaît par-dessus — comme des couches de peinture

Et les bonnes habitudes à garder :

- Définis le style (`fill`, `stroke`, `strokeWeight`) **juste avant** chaque forme
- Pense en **couches** : fond → éléments lointains → éléments proches

Dans le prochain chapitre, on va découvrir les **variables** — et ça va tout changer. Au lieu d'écrire des nombres fixes partout, on pourra utiliser des noms pour stocker et réutiliser des valeurs. C'est le premier pas vers des programmes vraiment dynamiques !

3.4 Les Variables

> 🎯 ***Objectif*** *: Comprendre ce qu'est une variable, savoir en créer, et les utiliser pour positionner et dimensionner des formes de manière intelligente. À la fin de ce chapitre, tu seras capable de dessiner des formes qui s'adaptent à la taille du canevas.*

C'est quoi une variable ?

Imagine une **boîte** avec :

- une **étiquette** (le nom de la variable)
- une **valeur** à l'intérieur (le contenu)
- un **type** inscrit sur la boîte (quel genre de contenu elle accepte)

```bash
┌──────────────┐
│     150      │  ← la valeur
└──────────────┘
  int posX       ← le type + le nom (l'étiquette)
```

En Processing, ça s'écrit :

```
int posX = 150;
```

Cette ligne fait trois choses d'un coup :

1. Elle crée une boîte de type `int` (nombre entier)
2. Elle colle l'étiquette `posX` dessus
3. Elle met la valeur `150` à l'intérieur

Le parallèle avec Scratch

En Scratch, tu crées une variable avec le bloc **"mettre maVariable à 10"**. En Processing, c'est la même idée, mais on écrit tout sur une seule ligne :

Scratch	Processing
Créer une variable "posX"	`int posX;`
Mettre "posX" à 150	`posX = 150;`
Créer + mettre en une fois	`int posX = 150;`
Ajouter 10 à "posX"	`posX = posX + 10;`

Les types de base

En Scratch, une variable peut contenir n'importe quoi : un nombre, du texte, un vrai/faux. En Processing, il faut **préciser le type** quand on crée la variable. C'est comme choisir la bonne taille de boîte avant d'y mettre quelque chose.

Pour l'instant, on va utiliser deux types :

Type	Ce que ça stocke	Exemple
`int`	Un nombre **entier** (sans virgule)	`int age = 25;`
`float`	Un nombre **à virgule**	`float prix = 9.99;`

```
int largeur = 200;       // ✅ Correct : 200 est un entier
float vitesse = 3.5;     // ✅ Correct : 3.5 a une virgule
int taille = 10.5;       // ❌ Erreur ! Un int ne peut pas stocker de virgule
```

💡 **Astuce** : en cas de doute, utilise `float`. Un `float` peut stocker un entier (comme `3.0`), mais un `int` ne peut **pas** stocker un nombre à virgule.

⚠️ **Attention** : en Processing (et en informatique en général), la virgule s'écrit avec un **point** ! On écrit `3.5` et pas `3,5`.

Modifier une variable

Une fois créée, tu peux **changer** la valeur d'une variable autant de fois que tu veux. Par contre, tu ne redis **pas** le type :

```
int score = 0;       // Création : on met le type
score = 10;          // Modification : PAS de type
score = score + 5;   // score vaut maintenant 15
```

⚠️ **Piège classique** : écrire `int score = 10;` puis plus loin `int score = 20;`. Processing va te dire que la variable existe déjà ! Le `int` ne se met qu'**une seule fois**, à la création.

Les opérations mathématiques

Les variables numériques supportent les opérations de base :

Opération	Symbole	Exemple	Résultat
Addition	`+`	`10 + 3`	`13`
Soustraction	`-`	`10 - 3`	`7`
Multiplication	`*`	`10 * 3`	`30`
Division	`/`	`10 / 3`	`3` ⚠️

⚠️ **Piège de la division entière** : quand tu divises deux `int`, le résultat est aussi un `int` — Processing **coupe** la partie décimale sans arrondir. `10 / 3` donne `3`, pas `3.33`.

Si tu veux un résultat précis, utilise des `float` :

```
int a = 10 / 3;       // a = 3 (partie décimale coupée !)
float b = 10.0 / 3.0; // b = 3.3333333
```

Tu peux bien sûr combiner variables et opérations :

```
int cote = 100;
int demiCote = cote / 2;   // 50
int doubleCote = cote * 2;  // 200
```

Les variables spéciales de Processing

Processing te fournit des variables déjà toutes faites, prêtes à l'emploi. Pas besoin de les créer, elles existent automatiquement :

`width` et `height` — les dimensions du canevas

Variable	Ce qu'elle contient
`width`	La **largeur** du canevas (en pixels)
`height`	La **hauteur** du canevas (en pixels)

Si tu as écrit `size(400, 300)`, alors `width` vaut `400` et `height` vaut `300`.

Pourquoi c'est utile ? Parce que ça te permet de **positionner des formes par rapport au canevas**, sans connaître sa taille à l'avance :

```
void setup() {
  size(400, 300);
}

void draw() {
  background(240);

  // Centre du canevas, quelle que soit la taille
  ellipse(width / 2, height / 2, 50, 50);
}
```

Essaie de changer `size(400, 300)` en `size(600, 400)` — le cercle reste **toujours au centre** ! Sans `width` et `height`, il faudrait recalculer à la main à chaque fois.

`mouseX` et `mouseY` — la position de la souris

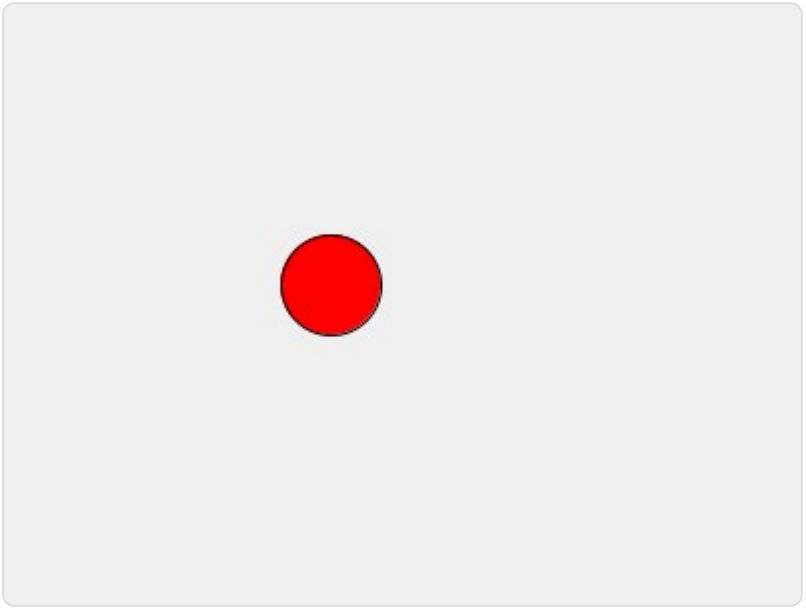
Variable	Ce qu'elle contient
`mouseX`	La position **horizontale** de la souris (en pixels depuis la gauche)
`mouseY`	La position **verticale** de la souris (en pixels depuis le haut)

Ces variables changent **60 fois par seconde** (à chaque passage dans `draw()`). C'est ce qui permet de créer de l'interactivité :

```
void setup() {
  size(400, 300);
}

void draw() {
  background(240);

  // Un cercle qui suit la souris
  fill(255, 0, 0);
  ellipse(mouseX, mouseY, 50, 50);
}
```



Essaie ce code ! Tu devrais voir un cercle rouge qui suit ta souris partout sur le canevas.

💡 **Expérience** : que se passe-t-il si tu **enlèves** la ligne `background(240)` ? Essaie, le résultat est surprenant ! (On reviendra sur cette astuce dans le chapitre sur l'animation.)

Bien nommer ses variables

Le nom d'une variable, c'est comme le nom d'un fichier sur ton ordinateur : il doit être **clair et descriptif**. Quelques règles :

Règle	✅ Bon	❌ Mauvais
Nom descriptif	`posX`, `largeurCarre`	`a`, `x1`, `truc`
Commence par une minuscule	`tailleCercle`	`TailleCercle`
Pas d'espaces ni d'accents	`couleurFond`	`couleur fond`, `côté`
Plusieurs mots → camelCase	`vitesseMax`	`vitesse_max`

La convention **camelCase** (première lettre en minuscule, puis majuscule à chaque nouveau mot) est le standard en Processing et en Java. Tu la retrouveras aussi en Arduino.

Exercice 3 — Carré centré

Consigne

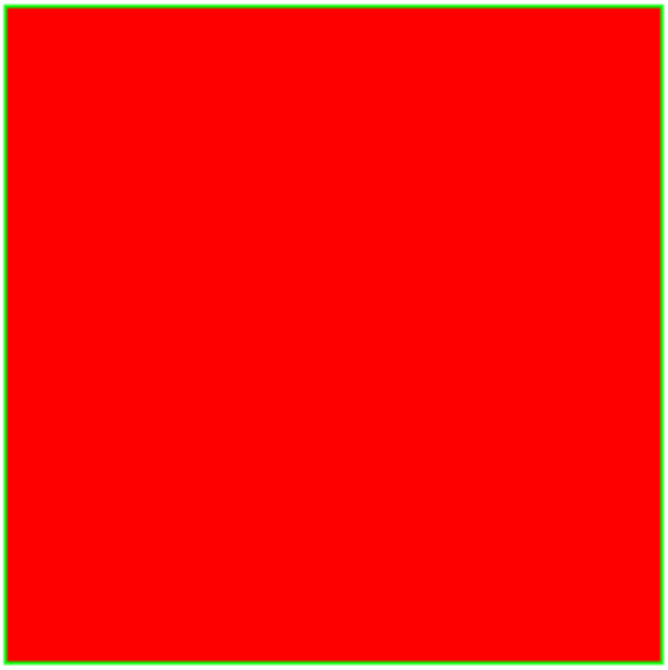
Dessine un **carré rouge** avec un **contour vert** parfaitement **au centre** de l'écran, en utilisant `width` et `height`.

Contraintes

- Le canevas fait `size(400, 300)`
- Le carré fait **100×100** pixels
- Le carré doit être **centré** sur le canevas
- Remplissage **rouge**, contour **vert**, épaisseur de contour de **3** pixels

Ce que tu devrais obtenir

Un carré rouge avec un contour vert, pile au milieu du canevas.



Indices

💡 **Indice 1** — Le centre du canevas est à la position `(width/2, height/2)`. Mais attention : par défaut, `rect()` dessine à partir du **coin supérieur gauche**, pas du centre !

💡 **Indice 2** — Tu as deux façons de centrer un rectangle :

- **Méthode calcul** : soustraire la moitié de la taille au point central → `rect(width/2 - 50, height/2 - 50, 100, 100)`
- **Méthode** `rectMode` : cherche `rectMode` dans la documentation de Processing (https://processing.org/reference). Il y a un mode qui change le comportement de `rect()` pour dessiner à partir du centre...

💡 **Indice 3** — `rectMode(CENTER)` fait en sorte que le premier paramètre de `rect()` soit le **centre** du rectangle, pas le coin. C'est beaucoup plus pratique pour centrer !

Exercice 4 — Dessiner dans le carré

Consigne

Reprends ton carré centré de l'exercice 3, et trace des **lignes à l'intérieur** en utilisant des variables pour tous les calculs de position.

Étape par étape :

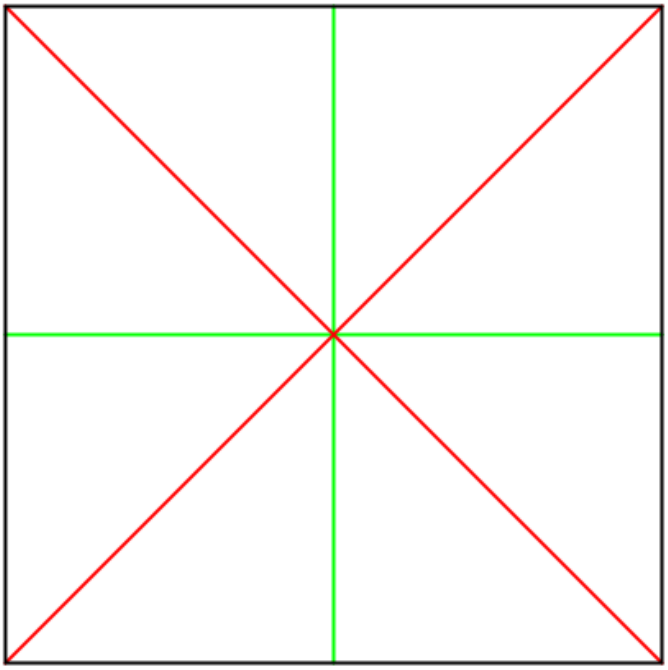
1. Crée des variables `x`, `y` pour la position du coin supérieur gauche du carré, `cote` pour la taille du côté, et `demiCote` pour la moitié
2. Dessine le carré en utilisant ces variables
3. Trace la **médiane verticale** (ligne du milieu haut au milieu bas du carré)
4. Trace la **médiane horizontale** (ligne du milieu gauche au milieu droit)
5. Trace les deux **diagonales** (coin à coin)

Contraintes

- Utilise **uniquement des variables** pour les positions (pas de nombres "en dur" sauf pour la taille initiale)
- Chaque ligne a une **couleur différente**
- Joue avec la **transparence** pour voir les lignes qui se croisent

Ce que tu devrais obtenir

Un carré avec 4 lignes colorées à l'intérieur : les deux médianes (en croix +) et les deux diagonales (en croix ×).



Indices

💡 **Indice 1** — Commence par créer tes variables. Si tu utilises `rectMode(CENTER)`, le coin supérieur gauche du carré est à `(width/2 - cote/2, height/2 - cote/2)`. Si tu préfères, tu peux aussi utiliser `rectMode(CORNER)` (le mode par défaut) et calculer `x` et `y` directement.

💡 **Indice 2** — La médiane verticale va du point `(x + demiCote, y)` au point `(x + demiCote, y + cote)`. La médiane horizontale va de `(x, y + demiCote)` à `(x + cote, y + demiCote)`.

💡 **Indice 3** — Les diagonales relient les coins opposés. La première va de `(x, y)` à `(x + cote, y + cote)`. La deuxième va de `(x + cote, y)` à `(x, y + cote)`.

🎯 Défi bonus — Le suiveur de souris

Reprends l'exercice 4, mais remplace les variables `x` et `y` par `mouseX` et `mouseY`. Ton carré (avec ses lignes intérieures) suivra la souris !

💡 **Astuce** : n'oublie pas le `background()` au début de `draw()`, sinon tu verras toutes les positions précédentes du carré.

Résumé

Concept	Syntaxe	Exemple
Créer une variable entière	`int nom = valeur;`	`int posX = 200;`
Créer une variable à virgule	`float nom = valeur;`	`float vitesse = 2.5;`
Modifier une variable	`nom = nouvelleValeur;`	`posX = posX + 10;`
Largeur du canevas	`width`	`ellipse(width/2, ...)`
Hauteur du canevas	`height`	`ellipse(..., height/2, ...)`
Position souris X	`mouseX`	`ellipse(mouseX, ...)`
Position souris Y	`mouseY`	`ellipse(..., mouseY, ...)`

Les points clés à retenir :

- Une variable a un **type**, un **nom** et une **valeur**
- Le type ne se déclare qu'**une seule fois** (à la création)
- `width`, `height`, `mouseX`, `mouseY` sont des variables **fournies par Processing**
- Utiliser des variables rend ton code **flexible** : change une valeur, tout s'adapte

Dans le prochain chapitre, on va découvrir les **conditions** — le moyen de faire réagir ton programme différemment selon la situation. C'est le `if` qui ouvre la porte à l'interactivité !

3.5 Les Conditions

> 🎯 ***Objectif*** *: Faire réagir ton programme différemment selon la situation grâce à* `if`*,* `else if` *et* `else`*. À la fin de ce chapitre, tu seras capable de créer un dessin interactif qui change en fonction de la position de la souris et des clics.*

C'est quoi une condition ?

Dans la vie de tous les jours, tu prends des décisions en permanence :

- **SI** il pleut **ALORS** je prends un parapluie, **SINON** je mets mes lunettes de soleil
- **SI** le feu est rouge **ALORS** je m'arrête
- **SI** j'ai faim **ALORS** je mange

En programmation, c'est exactement pareil. Une **condition** permet à ton programme de faire des choix : exécuter certaines instructions **uniquement si** quelque chose est vrai.

Le parallèle avec Scratch

En Scratch, tu connais le bloc **"Si ... Alors"**. En Processing, c'est le mot-clé `if` :

Scratch	Processing
Si <condition> Alors	`if (condition) { }`
Si <condition> Alors ... Sinon	`if (condition) { } else { }`

La logique est la même, seule l'écriture change.

La syntaxe du `if`

`if` seul — "si ... alors"

```
if (condition) {
  // Ce code s'exécute UNIQUEMENT si la condition est vraie
}
```

Exemple concret :

```
int temperature = 35;

if (temperature > 30) {
  fill(255, 0, 0);  // Rouge si c'est chaud
}
rect(50, 50, 100, 100);
```

Si `temperature` est supérieure à 30, le rectangle sera rouge. Sinon, il gardera la couleur par défaut (blanc).

`if ... else` — "si ... alors ... sinon"

```
if (condition) {
  // Exécuté si la condition est VRAIE
} else {
  // Exécuté si la condition est FAUSSE
}
```

```
int temperature = 15;

if (temperature > 30) {
  fill(255, 0, 0);  // Rouge si chaud
} else {
  fill(0, 0, 255);  // Bleu sinon
}
rect(50, 50, 100, 100);
```

C'est **l'un ou l'autre**, jamais les deux. Soit la condition est vraie et on passe dans le premier bloc, soit elle est fausse et on passe dans le `else`.

`if ... else if ... else` — plusieurs cas

Quand tu as plus de deux possibilités :

```
int temperature = 22;

if (temperature > 30) {
  fill(255, 0, 0);    // Rouge : chaud
} else if (temperature > 20) {
  fill(255, 165, 0);  // Orange : tiède
} else if (temperature > 10) {
  fill(0, 255, 0);    // Vert : frais
} else {
  fill(0, 0, 255);    // Bleu : froid
}
rect(50, 50, 100, 100);
```

Processing teste les conditions **de haut en bas** et s'arrête à la **première** qui est vraie. Si aucune n'est vraie, il passe dans le `else` (s'il y en a un).

⚠️ **Piège classique** : l'ordre des `else if` compte ! Si tu mets `temperature > 10` avant `temperature > 30`, la condition `> 10` sera vraie dès 11°C et Processing n'ira jamais tester `> 30`. Commence toujours par la condition la plus **restrictive**.

Les opérateurs de comparaison

Pour écrire des conditions, tu as besoin de **comparer** des valeurs :

Opérateur	Signification	Exemple	Vrai si...
`==`	Est égal à	`x == 10`	x vaut 10
`≠`	Est différent de	`x ≠ 10`	x ne vaut pas 10

Opérateur	Signification	Exemple	Vrai si...
`<`	Est plus petit que	`x < 10`	x est strictement inférieur à 10
`>`	Est plus grand que	`x > 10`	x est strictement supérieur à 10
`⩽`	Est plus petit ou égal	`x ⩽ 10`	x vaut 10 ou moins
`⩾`	Est plus grand ou égal	`x ⩾ 10`	x vaut 10 ou plus

⚠️ **ATTENTION** : pour comparer, c'est **deux** signes égal `==`. Un seul `=`, c'est pour **assigner** une valeur à une variable. C'est l'erreur n°1 des débutants !

```
int x = 10;      // ✅ Assigner : "x reçoit la valeur 10"
if (x == 10) {   // ✅ Comparer : "est-ce que x vaut 10 ?"
if (x = 10) {    // ❌ ERREUR ! Ça assigne au lieu de comparer
```

Combiner des conditions

Tu peux combiner plusieurs conditions avec les opérateurs logiques :

Opérateur	Signification	Exemple
`&&`	**ET** (les deux doivent être vraies)	`x > 0 && x < 100`
` ` `		` ` `
`!`	**NON** (inverse la condition)	`!mousePressed`

```
// Le clic est dans le quart supérieur gauche ?
if (mouseX < width/2 && mouseY < height/2) {
  fill(255, 0, 0);  // Rouge
}
```

C'est comme en français : "SI la souris est à gauche **ET** en haut **ALORS** c'est rouge."

Détecter la souris

Processing fournit des variables et fonctions spéciales pour l'interactivité :

Variable/Fonction	Ce qu'elle fait
`mouseX`	Position horizontale de la souris
`mouseY`	Position verticale de la souris

Variable/Fonction	Ce qu'elle fait
`mousePressed`	`true` si un bouton de la souris est enfoncé, `false` sinon
`mouseButton`	Quel bouton est enfoncé : `LEFT`, `RIGHT` ou `CENTER`

Exemple : changer de couleur au clic

```
void setup() {
  size(400, 300);
}

void draw() {
  background(240);

  if (mousePressed) {
    fill(255, 0, 0);    // Rouge quand on clique
  } else {
    fill(200);           // Gris sinon
  }

  ellipse(width/2, height/2, 150, 150);
}
```

Exemple : détecter gauche vs droite

```
void setup() {
  size(400, 300);
}

void draw() {
  if (mouseX < width / 2) {
    background(100, 150, 255);  // Bleu à gauche
  } else {
    background(255, 150, 100);  // Orange à droite
  }

  // Ligne de séparation
  stroke(255);
  strokeWeight(2);
  line(width/2, 0, width/2, height);
}
```

Essaie ce code : le fond change de couleur quand ta souris passe d'un côté à l'autre !

Quel bouton de souris ?

`mousePressed` détecte **tous** les boutons de la souris (gauche, droit, molette). Pour savoir **quel** bouton est enfoncé, on utilise la variable `mouseButton` :

Variable	Valeur	Signification
`mouseButton`	`LEFT`	Bouton gauche
`mouseButton`	`RIGHT`	Bouton droit
`mouseButton`	`CENTER`	Molette (clic du milieu)

On combine les deux avec `&&` :

```
if (mousePressed && mouseButton == LEFT) {
  // Le bouton GAUCHE est enfoncé
}
```

Exemple : dessiner au clic gauche

```
void setup() {
  size(400, 300);
  background(240);
}

void draw() {
  if (mousePressed && mouseButton == LEFT) {
    noStroke();
    fill(255, 0, 0);
    ellipse(mouseX, mouseY, 20, 20);
  }
}
```

Essaie ce code ! Tant que tu maintiens le clic gauche, tu dessines des cercles rouges comme un pinceau. `draw()` s'exécute 60 fois par seconde, donc le tracé est **fluide**.

💡 **Pourquoi pas de** `background()` dans `draw()` ? — Parce qu'on veut que les cercles **restent affichés**. Si on repeint le fond à chaque frame, les cercles précédents disparaissent. En mettant `background()` uniquement dans `setup()`, les dessins s'accumulent.

Exercice 5 — Dessin interactif par zones

Consigne

Le canevas est divisé en **4 zones** (quadrants). Quand on **maintient le clic gauche**, des cercles apparaissent à la position de la souris, mais leur **couleur dépend de la zone** :

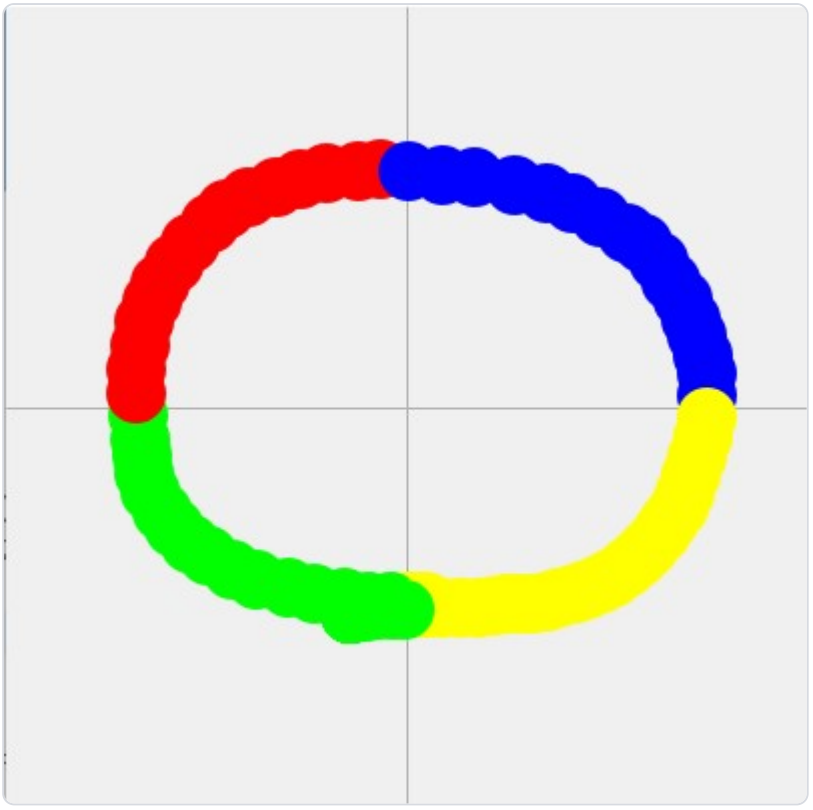
Zone	Position	Couleur
Haut-gauche	`mouseX < width/2` et `mouseY < height/2`	🟥 Rouge
Haut-droite	`mouseX ⩾ width/2` et `mouseY < height/2`	🟦 Bleu
Bas-gauche	`mouseX < width/2` et `mouseY ⩾ height/2`	🟩 Vert
Bas-droite	`mouseX ⩾ width/2` et `mouseY ⩾ height/2`	🟨 Jaune

Contraintes

- Canevas de `size(400, 400)`
- Le fond est dessiné **une seule fois** dans `setup()` (pas dans `draw()`) → les cercles restent visibles
- On dessine en continu avec `mousePressed && mouseButton == LEFT` dans `draw()`
- Chaque cercle fait **30 pixels** de diamètre
- Dessine les lignes de séparation des 4 zones (croix au centre de l'écran)

Ce que tu devrais obtenir

Un canevas divisé en 4 zones visibles. En maintenant le clic gauche et en déplaçant la souris, des cercles colorés apparaissent comme un pinceau, avec la couleur correspondant à la zone.



Indices

💡 **Indice 1** — Pour que les cercles restent affichés et ne disparaissent pas, **ne mets pas** `background()` dans `draw()`. Mets-le dans `setup()` à la place. Comme `draw()` ne repeint pas le fond, les cercles s'accumulent.

💡 **Indice 2** — Pour détecter le quadrant, combine deux conditions avec `&&` :

```
if (mouseX < width/2 && mouseY < height/2) {
  // Haut-gauche
}
```

💡 **Indice 3** — Structure globale du programme :

```
void setup() {
  size(400, 400);
  background(240);

  // Lignes de séparation
  stroke(180);
  line(width/2, 0, width/2, height);
  line(0, height/2, width, height/2);
}

void draw() {
  if (mousePressed && mouseButton == LEFT) {
    noStroke();

    // Détecte la zone et choisis la couleur
    if (mouseX < width/2 && mouseY < height/2) {
      fill(255, 0, 0);  // Rouge
    } else if (???) {
      // ...à toi de compléter !
    }

    ellipse(mouseX, mouseY, 30, 30);
  }
}
```

🎯 Défi bonus — La zone secrète

Ajoute une **5e zone circulaire** au centre de l'écran (un cercle de 100 pixels de diamètre). Si on dessine dans cette zone, les cercles sont **blancs**.

💡 **Astuce** : pour savoir si un point est dans un cercle, il faut calculer la **distance** entre le point (le clic) et le centre du cercle. Processing a une fonction `dist()` toute prête :

```
float d = dist(mouseX, mouseY, width/2, height/2);
if (d < 50) {
  // Le clic est dans la zone centrale (rayon = 50 pixels)
}
```

Pense à tester cette condition **en premier** (avant les quadrants), sinon les dessins au centre seront capturés par les quadrants avant d'arriver à la zone centrale !

Résumé

Concept	Syntaxe	Exemple
Si	`if (condition) { }`	`if (x > 10) { ... }`
Si ... sinon	`if (condition) { } else { }`	`if (x > 10) { ... } else { ... }`
Si ... sinon si ... sinon	`if ... else if ... else`	Tester plusieurs cas
Égal	`==`	`if (x == 10)`
Différent	`≠`	`if (x ≠ 10)`
Plus petit / plus grand	`<`, `>`, `⩽`, `⩾`	`if (x < 100)`
ET logique	`&&`	`if (x > 0 && x < 100)`
OU logique	`	
Souris enfoncée	`mousePressed`	`if (mousePressed) { }`
Bouton de souris	`mouseButton`	`if (mouseButton == LEFT) { }`
Distance	`dist(x1, y1, x2, y2)`	`dist(mouseX, mouseY, 200, 200)`

Scratch	Processing
Si <...> Alors	`if (condition) { }`
Si <...> Alors ... Sinon	`if (condition) { } else { }`
<souris pressée ?>	`mousePressed`
<touche ... pressée ?>	`mouseButton == LEFT`

Les points clés à retenir :

- `==` pour comparer, `=` pour assigner — ne les confonds pas !
- Teste les conditions de la plus **restrictive** à la plus **générale**
- `&&` = les deux conditions doivent être vraies, `||` = au moins une suffit
- `mousePressed` détecte le clic, `mouseButton` précise **quel** bouton (`LEFT`, `RIGHT`, `CENTER`)

Dans le prochain chapitre, on va découvrir les **boucles** — le moyen de répéter des instructions sans copier-coller, pour dessiner des motifs, des grilles et des dégradés en quelques lignes !

3.6 Les Boucles

> 🎯 ***Objectif*** *: Comprendre pourquoi et comment répéter des instructions automatiquement avec la boucle* `for`*. À la fin de ce chapitre, tu seras capable de dessiner des motifs répétitifs, des cercles concentriques et même un damier, en quelques lignes de code.*

Pourquoi des boucles ?

Imagine qu'on te demande de dessiner **10 cercles** les uns à côté des autres. Sans boucle, tu écrirais :

```
ellipse(30, 150, 40, 40);
ellipse(80, 150, 40, 40);
ellipse(130, 150, 40, 40);
ellipse(180, 150, 40, 40);
ellipse(230, 150, 40, 40);
ellipse(280, 150, 40, 40);
ellipse(330, 150, 40, 40);
ellipse(380, 150, 40, 40);
ellipse(430, 150, 40, 40);
ellipse(480, 150, 40, 40);
```

Dix lignes quasi identiques 😩 Et si on t'en demande 100 ? 1000 ? Tu vas copier-coller pendant des heures.

En programmation, **copier-coller, c'est un signal d'alarme**. Chaque fois que tu te retrouves à écrire plusieurs fois la même chose en changeant juste un nombre, c'est qu'il te faut une **boucle**.

Le parallèle avec Scratch

En Scratch, tu connais déjà le bloc **"Répéter 10 fois"**. En Processing, c'est la boucle `for` :

Scratch	Processing
🔄 Répéter **10** fois	`for (int i = 0; i < 10; i++)`

Le résultat est le même : le code à l'intérieur s'exécute 10 fois. Mais la version Processing est plus puissante, parce qu'elle te donne un **compteur** (`i`) qui te dit à quelle répétition tu en es.

Anatomie d'une boucle `for`

```
for (int i = 0; i < 10; i++) {
  // Ce code s'exécute 10 fois
  // i vaut 0, puis 1, puis 2, ... puis 9
}
```

Il y a trois parties séparées par des **points-virgules** :

```
for ( initialisation ; condition ; incrémentation )
        ↓                 ↓              ↓
    int i = 0        i < 10          i++
    "On commence     "Tant que       "À chaque tour,
     à 0"             i < 10,         on ajoute 1
                      on continue"    à i"
```

Partie	Ce qu'elle fait	Dans notre exemple
Initialisation	Crée le compteur et lui donne une valeur de départ	`int i = 0` → on commence à 0
Condition	Vérifie si on continue la boucle	`i < 10` → tant que i est plus petit que 10
Incrémentation	Modifie le compteur à chaque tour	`i++` → on ajoute 1 à i

💡 `i++` est un raccourci pour `i = i + 1`. C'est la même chose, juste plus court.

Ce qui se passe à chaque tour

Voici le déroulé complet d'un `for (int i = 0; i < 5; i++)` :

Tour	Valeur de `i`	Condition `i < 5` ?	Action
1	0	✅ Oui	Exécute le code
2	1	✅ Oui	Exécute le code
3	2	✅ Oui	Exécute le code
4	3	✅ Oui	Exécute le code
5	4	✅ Oui	Exécute le code
6	5	❌ Non	STOP, on sort de la boucle

La boucle s'exécute **5 fois** (pour i = 0, 1, 2, 3, 4). Quand `i` atteint 5, la condition `i < 5` est fausse → on s'arrête.

⚠️ **Piège classique** : la boucle commence à **0**, pas à 1. Et elle s'arrête **avant** d'atteindre la limite. `i < 5` donne les valeurs 0, 1, 2, 3, 4 → c'est bien **5 tours**, mais le dernier tour est à `i = 4`, pas `i = 5`.

La puissance du compteur `i`

Le compteur `i` n'est pas juste un numéro de tour : tu peux l'**utiliser dans tes calculs**. C'est ça qui rend la boucle `for` si puissante.

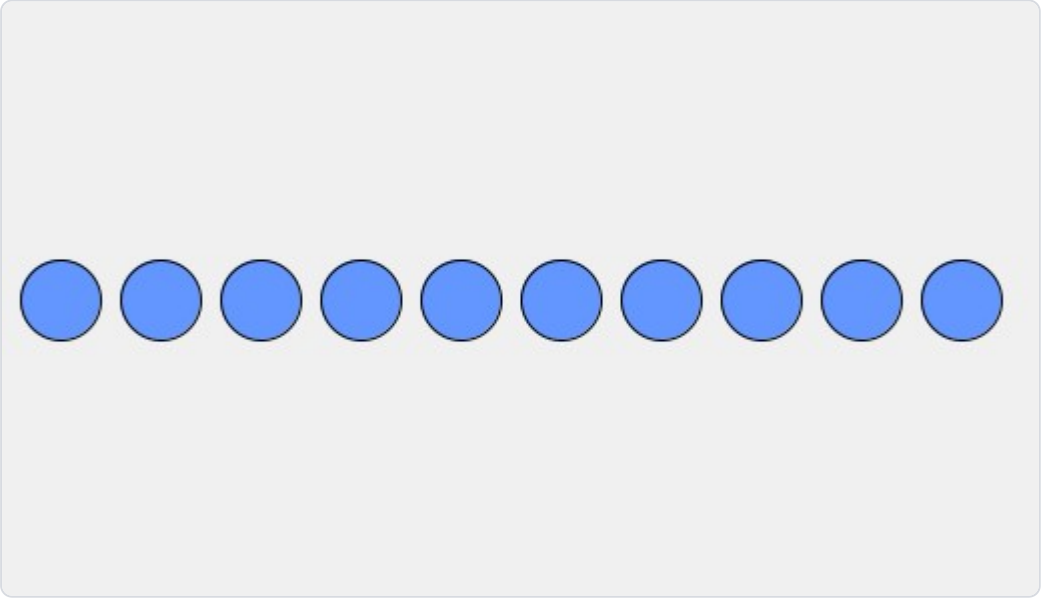
Reprenons nos 10 cercles du début :

```
void setup() {
  size(520, 300);
}

void draw() {
  background(240);

  fill(100, 150, 255);
  stroke(0);

  for (int i = 0; i < 10; i++) {
    ellipse(30 + i * 50, 150, 40, 40);
  }
}
```



Décortiquons `30 + i * 50` :

Tour	`i`	`30 + i * 50`	Position X
1	0	30 + 0 × 50	**30**

Tour	`i`	`30 + i * 50`	Position X
2	1	30 + 1 × 50	**80**
3	2	30 + 2 × 50	**130**
…	…	…	…
10	9	30 + 9 × 50	**480**

Deux lignes de code au lieu de dix. Et si tu veux 100 cercles ? Change juste `i < 10` en `i < 100` 😎

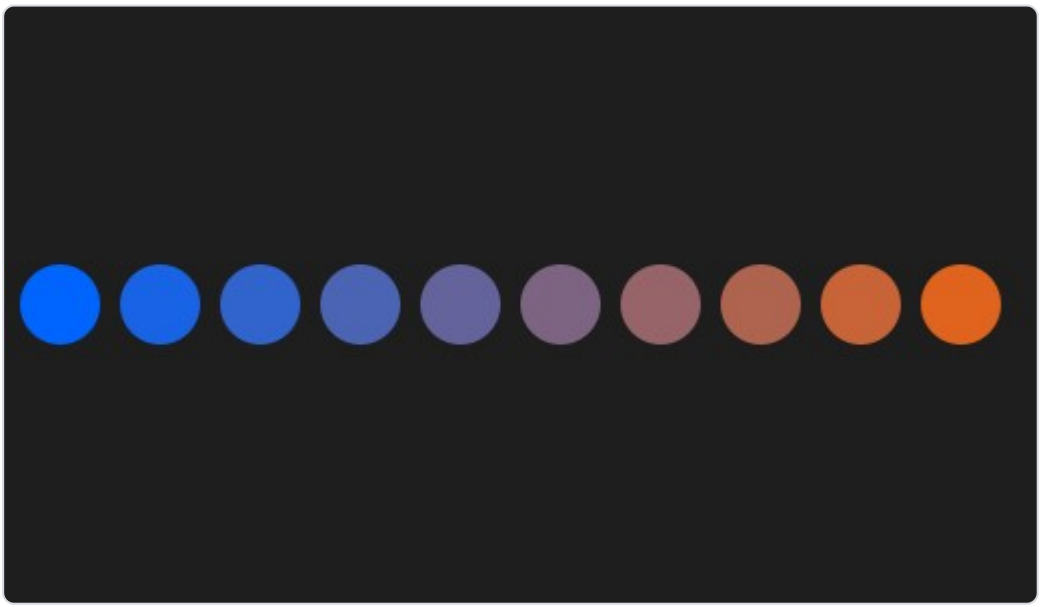
Utiliser `i` pour varier les couleurs

Le compteur `i` peut aussi servir à calculer des couleurs qui changent à chaque tour :

```
void setup() {
  size(520, 300);
}

void draw() {
  background(30);
  noStroke();

  for (int i = 0; i < 10; i++) {
    fill(i * 25, 100, 255 - i * 25);  // La couleur varie !
    ellipse(30 + i * 50, 150, 40, 40);
  }
}
```



Ici, `i * 25` fait varier la composante rouge de 0 à 225, et `255 - i * 25` fait varier le bleu en sens inverse. Résultat : un joli dégradé du bleu au rouge !

Utiliser une variable pour le nombre de répétitions

Plutôt que d'écrire un nombre "en dur" dans la boucle, utilise une **variable** :

```
int nbCercles = 10;

for (int i = 0; i < nbCercles; i++) {
  ellipse(30 + i * 50, 150, 40, 40);
}
```

L'avantage ? Tu changes **une seule valeur** et tout s'adapte. C'est exactement le même principe que les variables du chapitre précédent : rendre ton code **flexible**.

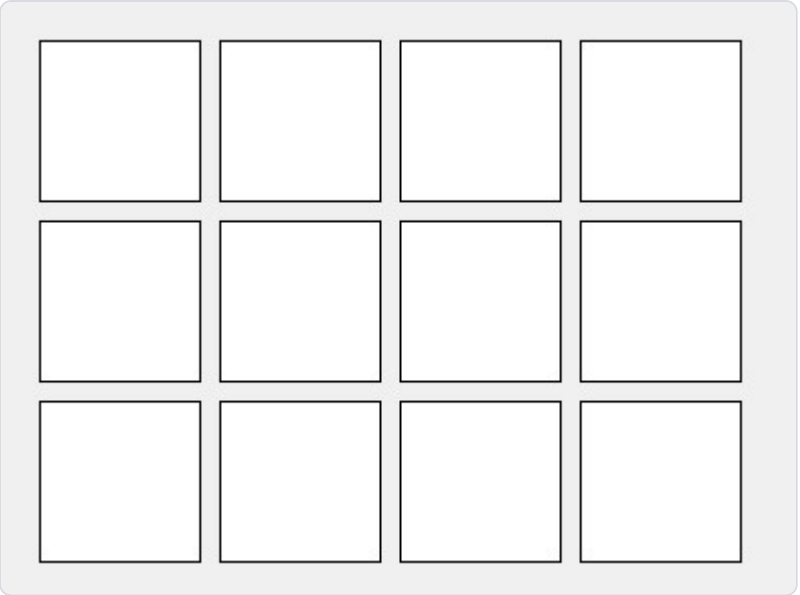
Les boucles imbriquées (grilles 2D)

Une boucle `for`, c'est une répétition en **une dimension** (une rangée). Deux boucles `for` imbriquées, c'est une répétition en **deux dimensions** (une grille).

```
void setup() {
  size(400, 300);
}

void draw() {
  background(240);

  for (int ligne = 0; ligne < 3; ligne++) {
    for (int colonne = 0; colonne < 4; colonne++) {
      rect(20 + colonne * 90, 20 + ligne * 90, 80, 80);
    }
  }
}
```



Comment ça marche ? La boucle extérieure (`ligne`) parcourt les lignes, et pour **chaque ligne**, la boucle intérieure (`colonne`) dessine tous les éléments de cette ligne :

```
ligne=0 : colonne=0, colonne=1, colonne=2, colonne=3   → 1ère rangée
ligne=1 : colonne=0, colonne=1, colonne=2, colonne=3   → 2ème rangée
ligne=2 : colonne=0, colonne=1, colonne=2, colonne=3   → 3ème rangée
```

C'est comme lire un livre : on va de gauche à droite (colonnes), puis on passe à la ligne suivante.

💡 **Astuce** : nomme tes compteurs `ligne` et `colonne` (ou `row` et `col`) plutôt que `i` et `j` . C'est plus clair quand on relit le code.

Exercice 6 — L'arc-en-ciel (sans boucle)

Consigne

Dessine un **arc-en-ciel** : des demi-cercles concentriques de couleurs différentes. Le centre de l'arc-en-ciel est en **bas du canevas**. Ajoute un petit paysage autour (sol vert, ciel bleu, soleil...).

Les couleurs de l'arc-en-ciel, du plus grand au plus petit : rouge, orange, jaune, vert, bleu, indigo, violet.

Contraintes

- Canevas de `size(500, 400)`
- Utilise `arc()` avec les angles `0` et `PI` (demi-cercle supérieur) — ou bien des `ellipse()` que tu empiles
- **Pas de boucle** pour cet exercice (on fera la version boucle dans l'exercice suivant)

- `noStroke()` pour éviter les contours entre les arcs

Ce que tu devrais obtenir

Un arc-en-ciel avec 7 bandes de couleurs au-dessus d'un paysage simple.



Indices

💡 **Indice 1** — L'astuce est de dessiner des **ellipses** (ou arcs) du plus grand au plus petit, comme des poupées russes. La plus grande ellipse (rouge) est dessinée en premier, puis l'orange par-dessus (un peu plus petite), etc. La dernière ellipse (violette) est la plus petite.

💡 **Indice 2** — Centre tes demi-cercles en bas du canevas, par exemple à la position `(250, 400)`. Commence avec un diamètre de 400, puis diminue de 50 à chaque bande.

💡 **Indice 3** — Tu peux utiliser `arc(250, 400, 400, 400, PI, TWO_PI)` pour un demi-cercle vers le haut. Ou, plus simple : dessine des `ellipse()` complètes et cache le bas avec le rectangle du sol !

Exercice 7 — Cercles concentriques (avec boucle)

Consigne

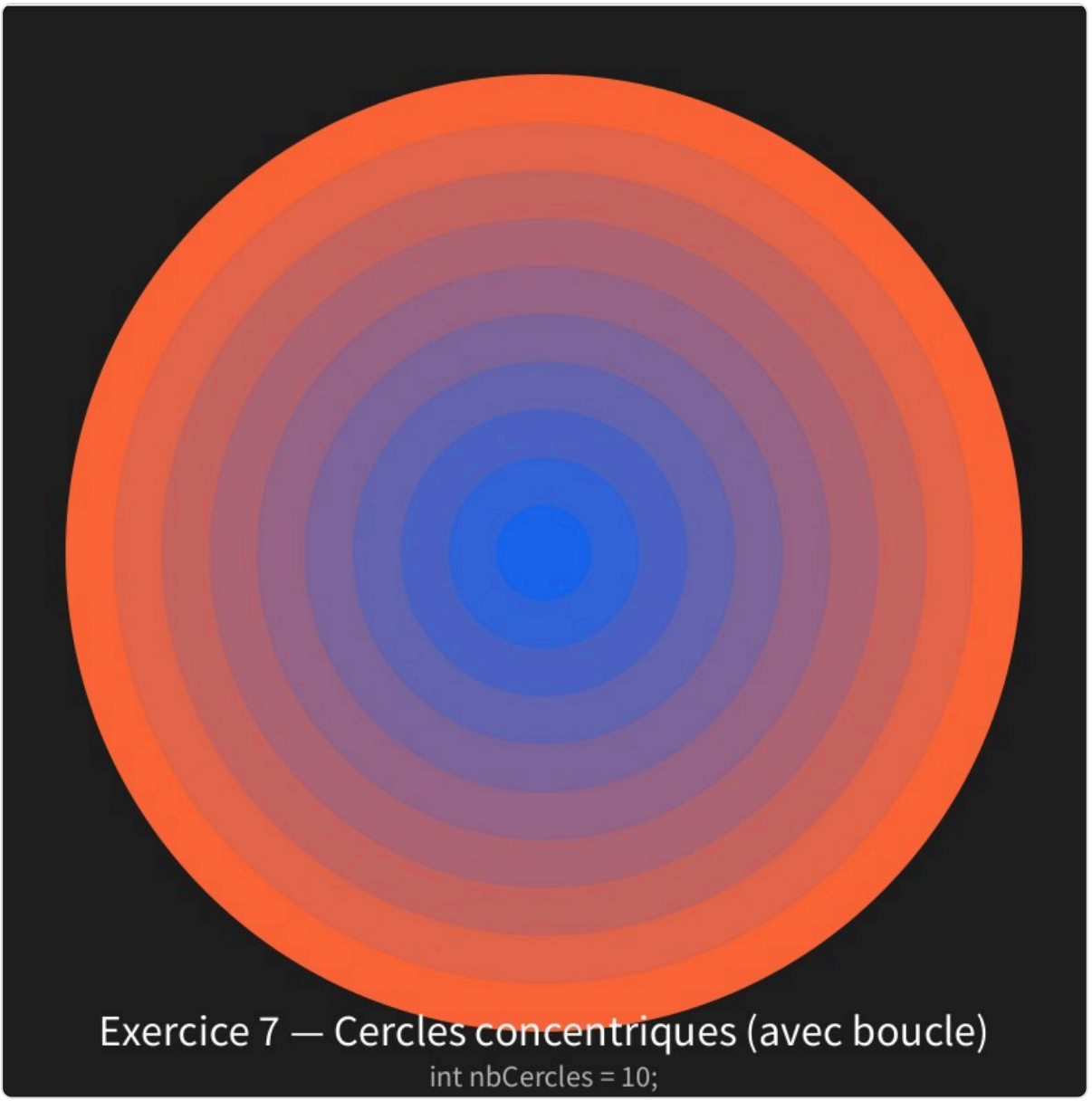
Reprends le principe des cercles concentriques, mais cette fois avec une **boucle** `for`. Le nombre de cercles est stocké dans une variable `nbCercles`. Chaque cercle a une **couleur différente** calculée automatiquement à partir du compteur `i`.

Contraintes

- Canevas de `size(400, 400)`
- Les cercles sont **centrés** au milieu du canevas
- Le nombre de cercles est dans une variable (commence avec `nbCercles = 8`)
- Chaque cercle a une couleur calculée avec `i` (pas de couleurs écrites à la main)
- `noStroke()` pour un rendu propre

Ce que tu devrais obtenir

Des cercles concentriques avec un dégradé de couleurs automatique, du plus grand (extérieur) au plus petit (intérieur).



Indices

💡 **Indice 1** — Attention à l'**ordre** ! Il faut dessiner du **plus grand au plus petit** (sinon les grands cercles cachent les petits). Astuce : fais ta boucle à l'envers avec `for (int i = nbCercles; i > 0; i--)`, ou bien calcule le rayon en partant du plus grand.

💡 **Indice 2** — Le diamètre du cercle `i` peut être `i * 40`. La couleur peut varier avec `fill(i * 30, 100, 255 - i * 30)`.

💡 **Indice 3** — Pour le bonus : remplace `nbCercles = 8` par `nbCercles = mouseX / 20`. Le nombre de cercles changera en fonction de la position horizontale de la souris !

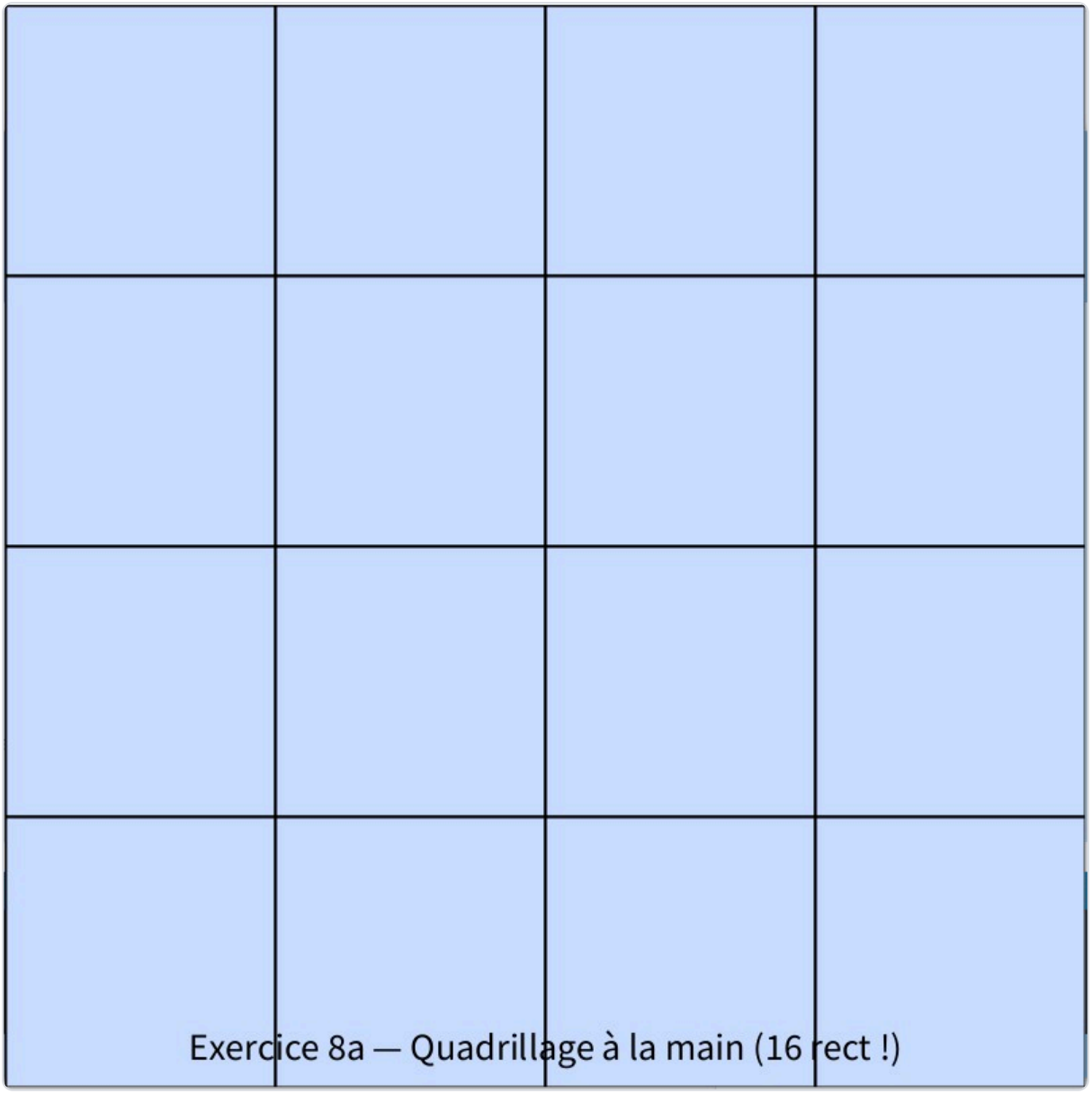
Exercice 8 — Le damier (boucles imbriquées)

Cet exercice se fait en **trois étapes progressives**.

Étape 1 — Quadrillage à la main (sans boucle)

Dessine un quadrillage de **4×4 carrés** de 100×100 pixels, tous de la même couleur. Oui, ça va être long et répétitif — c'est fait exprès 😉

Faites des copier-coller pour aller plus vite !



Étape 2 — Quadrillage avec boucles

Remplace tes 16 lignes de `rect()` par **2 boucles** `for` imbriquées. Le nombre de cases par côté est dans une variable `nbCases`.

Étape 3 — Damier

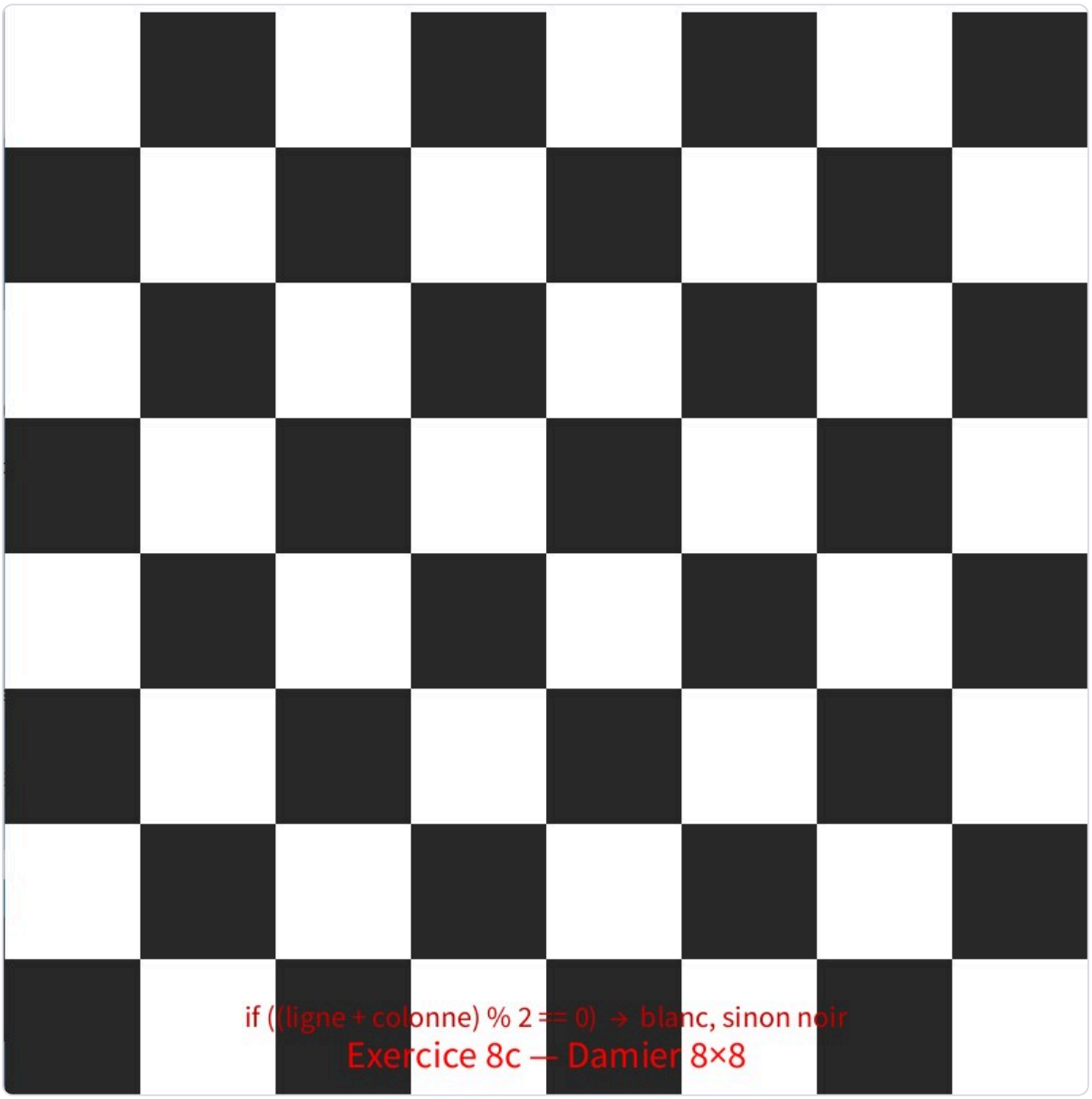
Alterne les couleurs **noir et blanc** comme un échiquier.

Contraintes

- Canevas de `size(400, 400)` (pour que les 4×4 carrés de 100px remplissent tout l'écran)
- Pour l'étape 2 : la taille des cases se calcule avec `width / nbCases`
- Pour l'étape 3 : utilise une condition `if` pour choisir la couleur

Ce que tu devrais obtenir

Étape 1 → une grille de carrés identiques. Étape 2 → la même grille, mais en 4 lignes de code. Étape 3 → un échiquier noir et blanc.



Indices

💡 **Indice 1 (étape 2)** — La position X d'une case est `colonne * taille` et la position Y est `ligne * taille`, avec `taille = width / nbCases`.

💡 **Indice 2 (étape 3)** — L'astuce du damier repose sur une observation mathématique : si la **somme** du numéro de ligne et du numéro de colonne est **paire**, la case est blanche ; si elle est **impaire**, la case est noire.

💡 **Indice 3 (étape 3)** — Pour savoir si un nombre est pair, on utilise l'opérateur **modulo** `%` (le reste de la division). `(ligne + colonne) % 2` donne `0` si la somme est paire, `1` si elle est impaire :

```
if ((ligne + colonne) % 2 == 0) {
  fill(255);  // Blanc
} else {
  fill(0);    // Noir
}
```

Résumé

Concept	Syntaxe	Exemple
Boucle for	`for (int i = 0; i < n; i++)`	Répéter n fois
Compteur	`i`	Vaut 0, 1, 2, ... n-1
Incrémenter	`i++`	Raccourci pour `i = i + 1`
Position calculée	`debut + i * espacement`	`30 + i * 50`
Couleur calculée	`i * facteur`	`fill(i * 25, 0, 0)`
Boucles imbriquées	2 `for` = grille 2D	`for ligne { for colonne }`
Modulo	`a % b`	Reste de la division → pair/impair

Scratch	Processing
🔄 Répéter 10 fois	`for (int i = 0; i < 10; i++)`
Variable "compteur" dans la boucle	`i` (automatique)

Les points clés à retenir :

- Une boucle **évite le copier-coller** et rend ton code adaptable
- Le compteur `i` est une **variable gratuite** que tu peux utiliser dans tes calculs
- Deux boucles imbriquées = une **grille** (lignes × colonnes)
- Mets le nombre de répétitions dans une **variable** pour pouvoir le changer facilement

Dans le prochain chapitre, on va découvrir les **fonctions** — le moyen de créer tes propres blocs réutilisables, comme les "Mes Blocs" de Scratch !

CHAPITRE 4

Bienvenue dans le monde de l'électronique et d'Arduino

> 🎯 ***Objectif de ce chapitre*** *: Comprendre ce qu'est Arduino, découvrir la plateforme Électro-Lab et avoir une vue d'ensemble du parcours qui vous attend. Durée : ~15 min de lecture.*

Slides du chapitre

- Arduino.pdf

De Scratch à Arduino : le chemin parcouru

Depuis le début de ce cours, vous avez déjà accompli un sacré bout de chemin. Faisons le point :

Étape	Ce que vous avez appris	Le lien avec Arduino
Scratch	Les blocs, les variables, les boucles, les conditions	Arduino utilise les mêmes concepts… mais en texte
Processing	Écrire du vrai code (syntaxe, points-virgules, accolades)	Le langage Arduino ressemble énormément à Processing
LEGO EV3	Programmer un robot physique avec des capteurs et moteurs	Arduino fait exactement pareil, mais avec de vrais composants électroniques

Chacune de ces étapes vous a préparé, morceau par morceau, à ce qui vient maintenant. Vous ne partez pas de zéro — loin de là.

💡 **Le déclic** : Avec Scratch, vous donniez des ordres à un chat sur un écran. Avec EV3, vous donniez des ordres à un robot en briques. Avec Arduino, vous allez donner des ordres à… tout ce que vous pouvez imaginer. Une lampe, un capteur de température, un robot roulant, une station météo, un système d'alarme. La seule limite, c'est votre créativité (et le nombre de fils sur votre table).

C'est quoi, un Arduino ?

Un Arduino, c'est un **mini-ordinateur spécialisé**. Contrairement à votre PC qui fait tourner Windows, des navigateurs et des jeux, un Arduino ne fait qu'une seule chose : **exécuter le programme que vous lui donnez**, encore et encore, sans jamais s'arrêter.

Imaginez un employé très obéissant mais très littéral. Il fait exactement ce que vous lui dites, dans l'ordre que vous lui dites, sans jamais prendre d'initiative. Si vous lui dites "allume la lumière, attends 1 seconde, éteins la lumière, attends 1 seconde", il le fera à l'infini. Si vous oubliez de lui dire "éteins la lumière"... elle restera allumée pour toujours.

L'Arduino UNO — votre nouveau meilleur ami

L'Arduino UNO est la carte que vous allez utiliser. Voici ce qu'il y a dessus :

Élément	Rôle	Analogie
Microcontrôleur (la puce noire au centre)	Le "cerveau" qui exécute votre programme	Le processeur de votre PC
Pins digitales (0 à 13)	Envoyer ou recevoir du courant (ON/OFF)	Des interrupteurs
Pins analogiques (A0 à A5)	Lire des valeurs progressives (0 à 1023)	Un thermomètre gradué
Pin 5V et 3.3V	Fournir du courant aux composants	Les prises de courant
Pin GND	Le retour du courant (la "masse")	Le tuyau de retour de l'eau
Port USB	Connecter au PC pour programmer	Le câble réseau
LED intégrée (pin 13)	Un petit voyant pour tester rapidement	Votre premier cobaye

⚠️ **Important** : Les pins de l'Arduino ne fournissent que très peu de courant (40 mA par pin). C'est suffisant pour allumer une LED, mais pas pour faire tourner un moteur directement. On verra plus tard comment résoudre ça.

Pourquoi Arduino et pas autre chose ?

Il existe des dizaines de cartes de développement (Raspberry Pi, ESP32, STM32, Micro:bit...). Alors pourquoi commencer par Arduino ?

- **Simplicité** : on branche, on code, ça marche. Pas de système d'exploitation à configurer.
- **Communauté** : des millions de tutoriels, projets, et forums d'entraide dans toutes les langues.
- **Robustesse** : c'est très difficile de "casser" un Arduino en faisant une erreur de programmation. Au pire, on recharge un nouveau programme.
- **Prix** : une carte compatible coûte moins de 5€.
- **Progressivité** : une fois à l'aise avec Arduino, passer à l'ESP32 (WiFi, Bluetooth) ou au Raspberry Pi devient naturel.

💡 **Fun fact** : Arduino a été créé en 2005 en Italie, à l'Interaction Design Institute d'Ivrea. Le nom vient d'un bar local, le "Bar di Re Arduino", lui-même nommé d'après un roi italien du XIe siècle. Oui, votre carte porte le nom d'un bistrot. 🍺

Arduino dans la vraie vie

Arduino n'est pas qu'un jouet pour cours d'informatique. Voici quelques exemples concrets de ce que des gens fabriquent avec :

Domaine	Exemple	Composants utilisés
Domotique	Système d'arrosage automatique qui mesure l'humidité du sol	Arduino + capteur d'humidité + pompe
Art interactif	Installation lumineuse qui réagit aux mouvements	Arduino + capteur PIR + LEDs NeoPixel
Météo	Station météo qui affiche température, humidité et pression	Arduino + DHT22 + BME280 + écran OLED
Robotique	Robot éviteur d'obstacles	Arduino + moteurs + capteur ultrason
Musique	Boîte à musique qui joue des mélodies	Arduino + buzzer piézo
Sécurité	Détecteur d'intrusion avec notification	Arduino + capteur PIR + buzzer

Et devinez quoi ? Vous allez réaliser plusieurs de ces projets durant ce cours.

Ce que vous allez apprendre

Le parcours est divisé en **4 grandes unités**, pour un total de **14 chapitres**. Voici la vue d'ensemble :

Unité 1 — Les fondations de l'électricité (Chapitres 1 à 4)

Avant de toucher à l'Arduino, il faut comprendre comment le courant circule. C'est comme apprendre le code de la route avant de conduire.

- **C'est quoi l'électricité ?** — Tension, courant, résistance avec l'analogie de l'eau
- **La loi d'Ohm** — La formule magique V = R × I
- **Les composants** — LED, résistance, condensateur, transistor, bouton
- **Le breadboard et les schémas** — Lire un plan et monter un circuit

Unité 2 — Premiers pas avec Arduino (Chapitres 5 à 8)

On branche, on code, on fait clignoter des trucs. Les mains dans le cambouis.

- **Arduino : le cerveau** — Anatomie de la carte, installation de l'IDE, premier programme Blink
- **LED externe** — Monter un vrai circuit sur breadboard, feu tricolore
- **Boutons et interactions** — Lire un bouton, anti-rebond, toggle
- **PWM et fade** — Faire varier la luminosité d'une LED (pas juste ON/OFF)

Unité 3 — Capteurs et afficheurs (Chapitres 9 à 11)

On apprend à "sentir" le monde réel et à afficher des informations.

- **Capteurs analogiques** — Photorésistance, potentiomètre, température
- **Afficheurs** — Écran LCD, OLED, afficheur 7 segments
- **Communication série** — Parler avec le PC via le Serial Monitor

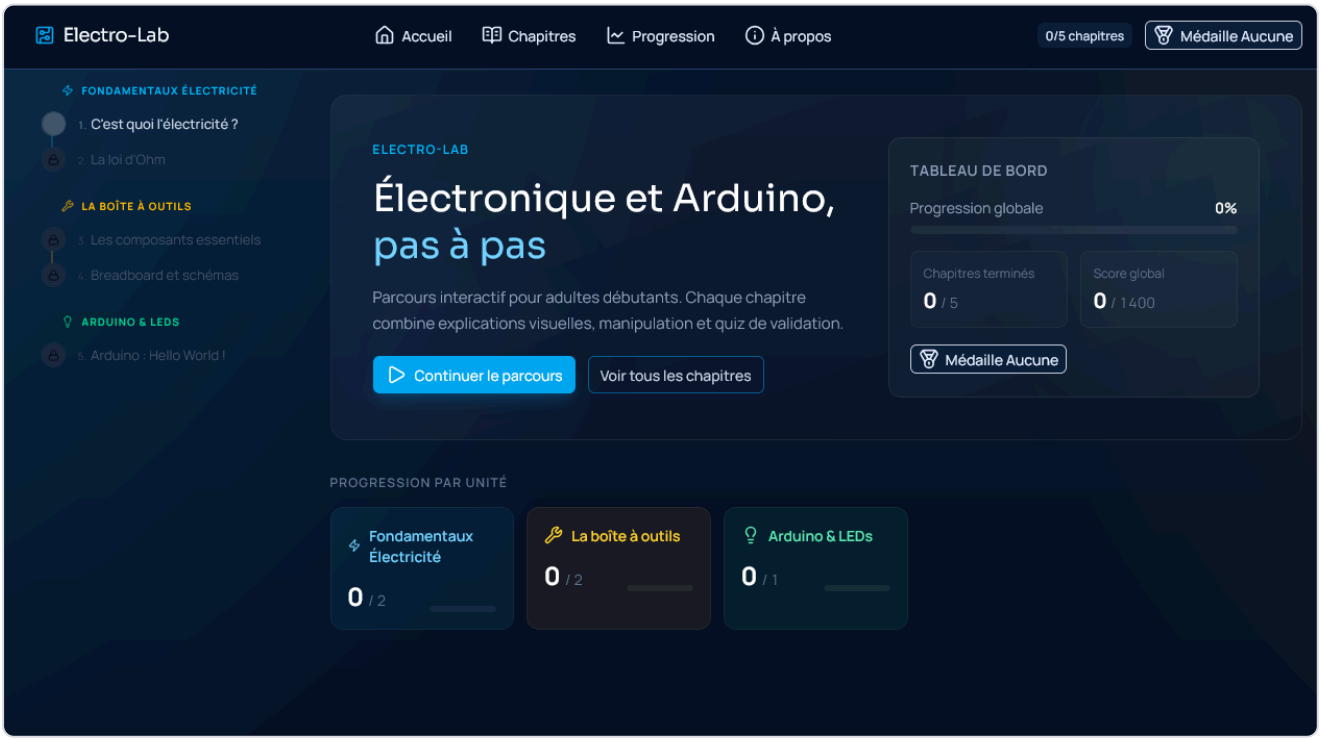
Unité 4 — Projets intégrés (Chapitres 12 à 14)

On combine tout pour construire de vrais projets complets.

- **Station météo** — Température, humidité, pression, affichage OLED
- **Robot 2WD** — Châssis, moteurs, évitement d'obstacles
- **Projet libre** — Votre idée, vos composants, votre créativité

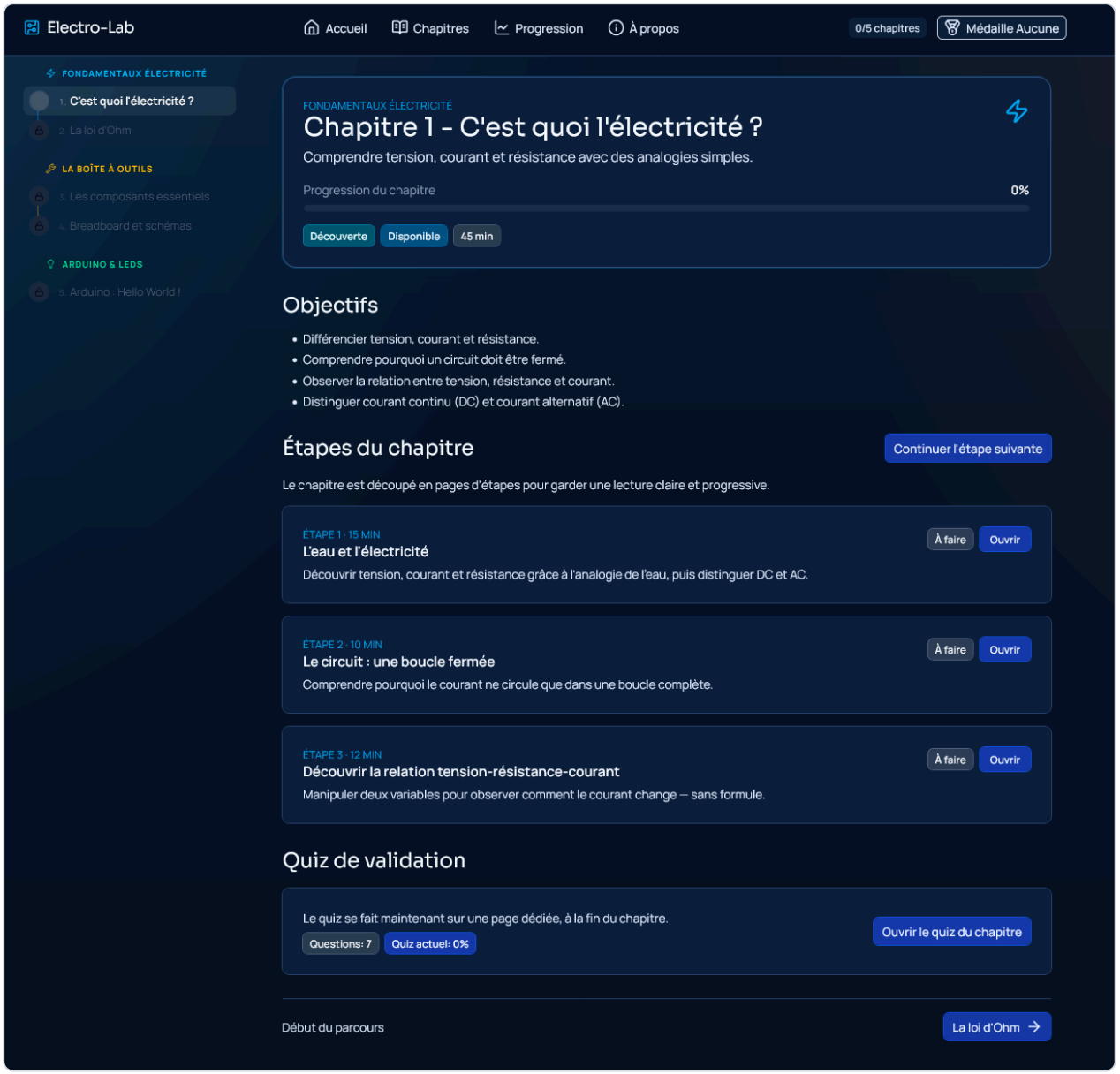
Votre outil d'apprentissage : Électro-Lab

Pour vous accompagner tout au long de ce parcours, on a développé une plateforme web dédiée : **Électro-Lab**.



Qu'est-ce qu'Électro-Lab ?

Électro-Lab, c'est votre support de cours interactif. Plutôt qu'un PDF statique de 200 pages, c'est une application web où chaque chapitre combine des explications visuelles, des manipulations interactives et des quiz de validation.



Comment ça fonctionne ?

Chaque chapitre suit le même schéma en 4 temps :

Étape	Ce que vous faites	Durée
1. Découverte	Vous lisez le contenu, manipulez les widgets interactifs, explorez les analogies	10-15 min
2. Manipulation	Vous montez le circuit sur breadboard en suivant le guide pas à pas	15-30 min

Étape	Ce que vous faites	Durée
3. Programmation	Vous écrivez (ou complétez) le code Arduino et le téléversez	10-20 min
4. Quiz	Vous répondez à un quiz de validation pour débloquer la suite	5-10 min

Les widgets interactifs

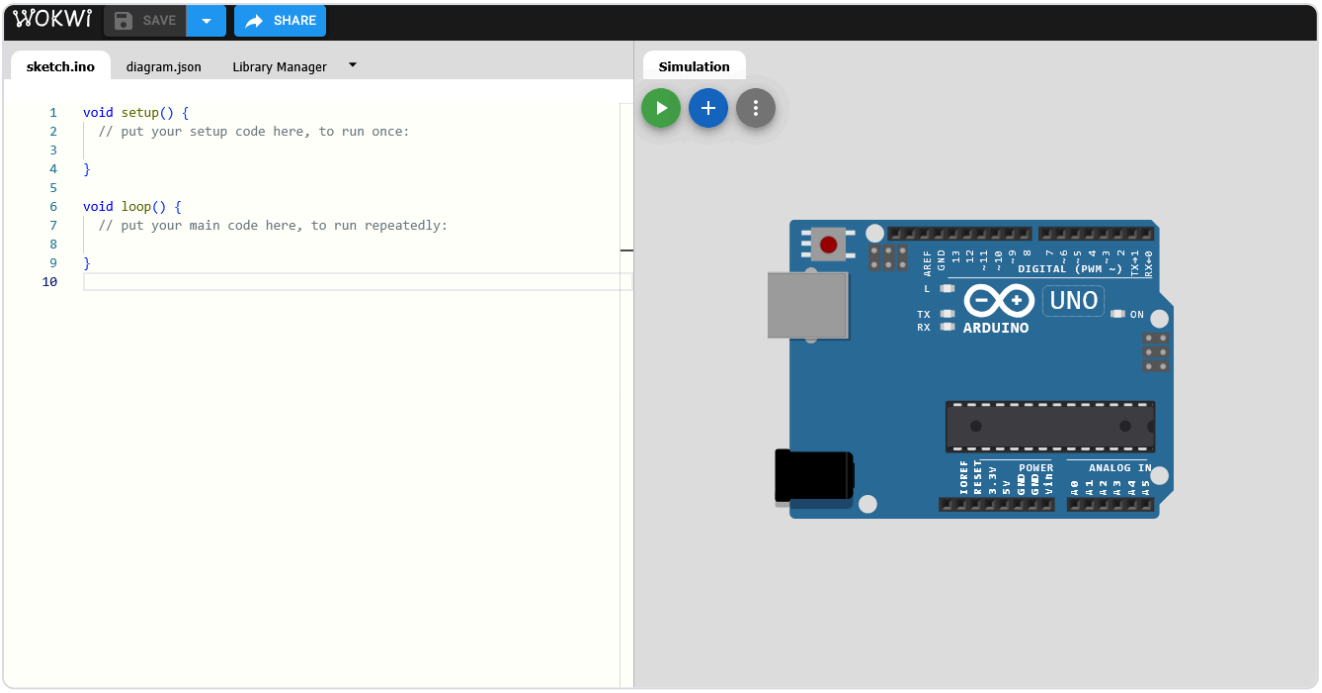
C'est là qu'Électro-Lab se distingue d'un simple cours en ligne. Vous ne faites pas que lire — vous manipulez :

- **Sliders de découverte** : bougez des curseurs pour observer comment la tension et la résistance affectent le courant, sans formule
- **Schémas de montage** : suivez le circuit pas à pas avec des images zoomables de l'Arduino et du breadboard
- **Décodeur de résistances** : cliquez sur les bandes de couleur pour lire la valeur d'une résistance
- **Éditeur de code** : complétez les blancs dans un programme Arduino pour valider votre compréhension
- **Simulations Wokwi** : testez vos circuits en ligne avant de les monter physiquement

Exemple de widget



Wokwi (Simulateur)



Progression et médailles

Votre progression est sauvegardée automatiquement. Chaque chapitre vous rapporte des points, et votre score global détermine votre médaille :

Médaille	Seuil	Ce que ça signifie
🥉 **Bronze**	40%	Vous avez compris les bases
🥈 **Argent**	60%	Vous maîtrisez bien le sujet
🥇 **Or**	80%	Vous êtes à l'aise pour aller plus loin
💎 **Diamant**	95%	Vous êtes un as de l'électronique

💡 **Pas de stress** : les quiz peuvent être refaits autant de fois que vous voulez. L'objectif n'est pas de "réussir du premier coup", c'est de comprendre. Si vous vous trompez, le quiz vous aide à identifier ce qui n'est pas clair.

Accès à Électro-Lab

La plateforme est accessible depuis n'importe quel navigateur (ordinateur, tablette, téléphone). Vous pouvez l'utiliser en classe sur les PC de l'école, mais aussi chez vous pour revoir les concepts à votre rythme.

Le matériel que vous allez utiliser

Tout le matériel est fourni en classe. Voici ce que vous trouverez sur votre table :

Composant	Quantité	À quoi ça sert
Arduino UNO	1 par binôme	Le cerveau de vos montages
Breadboard	1 par binôme	La plaque de montage sans soudure
Câbles Dupont	Plein !	Relier les composants entre eux
LEDs (rouge, verte, jaune, bleue)	Plusieurs de chaque	Vos premiers indicateurs visuels
Résistances (220Ω, 1kΩ, 10kΩ)	Plusieurs de chaque	Protéger les LEDs et gérer le courant
Boutons poussoirs	2-3 par binôme	Créer des interactions
Potentiomètre	1 par binôme	Varier une valeur progressivement
Câble USB	1 par binôme	Brancher l'Arduino au PC

⚠️ **Règles d'or du matériel** :

1. **Ne jamais brancher un composant quand l'Arduino est alimenté.** Toujours débrancher l'USB d'abord.
2. **Une LED sans résistance = une LED morte.** Toujours mettre une résistance en série.
3. **Les fils ont des couleurs pour une raison** : rouge = alimentation (+), noir = masse (GND), autres = signaux. Respectez cette convention, ça vous évitera bien des galères.

Et la programmation dans tout ça ?

Le langage Arduino est basé sur le **C/C++**, mais en version simplifiée. Si vous avez survécu à Processing (et vous avez survécu, bravo 👏), vous allez retrouver beaucoup de choses familières :

Processing	Arduino	C'est pareil ?
`void setup() { }`	`void setup() { }`	✅ Identique !
`void draw() { }`	`void loop() { }`	✅ Même principe (boucle infinie)
`int x = 10;`	`int x = 10;`	✅ Identique !
`if (x > 5) { }`	`if (x > 5) { }`	✅ Identique !
`for (int i = 0; i < 10; i++)`	`for (int i = 0; i < 10; i++)`	✅ Identique !
`ellipse(100, 100, 50, 50);`	`digitalWrite(13, HIGH);`	🔄 Fonctions différentes, même syntaxe

La grosse différence : en Processing, vous dessiniez sur un écran. En Arduino, vous contrôlez des composants physiques. Mais la structure du code, la logique, la syntaxe — c'est la même chose.

Ce qui vous attend : les projets

Voici un aperçu des projets concrets que vous réaliserez au fil du parcours :

🚦 Le feu tricolore

Votre premier "vrai" projet : trois LEDs qui s'allument dans l'ordre rouge → orange → vert, comme un feu de signalisation. Simple, mais c'est le premier circuit que vous programmerez de A à Z.

🌡️ La station météo

Un capteur de température et d'humidité (DHT22) relié à un écran OLED qui affiche les mesures en temps réel. Vous pourrez même y ajouter un capteur de pression atmosphérique (BME280) pour prédire la météo.

🤖 Le robot éviteur d'obstacles

Un châssis 2 roues motrices, un capteur ultrason monté sur un servomoteur qui "scanne" l'environnement, et un programme qui fait tourner le robot quand il détecte un mur. C'est le projet final qui combine tout ce que vous aurez appris.

Récapitulatif

Avant de plonger dans Électro-Lab pour le premier chapitre, retenez ces quelques points :

- ✅ **Vous n'êtes pas des débutants.** Scratch, Processing et EV3 vous ont donné toutes les bases en programmation. Arduino, c'est la suite logique.
- ✅ **Arduino = un mini-ordinateur spécialisé** qui exécute votre programme en boucle et contrôle des composants physiques.
- ✅ **Électro-Lab est votre guide.** Chaque chapitre mélange théorie interactive, montage guidé, code et quiz. Suivez les étapes, à votre rythme.
- ✅ **Le matériel est fourni.** Vous travaillez en binômes, et tout est sur votre table.
- ✅ **Le langage Arduino, c'est du Processing** (ou presque). Vous connaissez déjà la syntaxe.

💡 **Prochain chapitre sur Électro-Lab** : "C'est quoi l'électricité ?" — Vous allez découvrir la tension, le courant et la résistance grâce à l'analogie de l'eau. Aucune formule, juste de la logique et des sliders interactifs.

Liens utiles

- https://electrolab.choboai.com
- https://wokwi.com/projects/new/arduino-uno

- https://raw.githubusercontent.com/liffiton/Arduino-Cheat-Sheet/refs/heads/master/Arduino Cheat Sheet.pdf

4.1 L'IDE Arduino — Installer et lancer son premier programme

> 🎯 ***Objectif*** *: Installer l'IDE Arduino, comprendre son interface, et être prêt à écrire votre premier programme. Durée : ~10 min.*

C'est quoi l'IDE Arduino ?

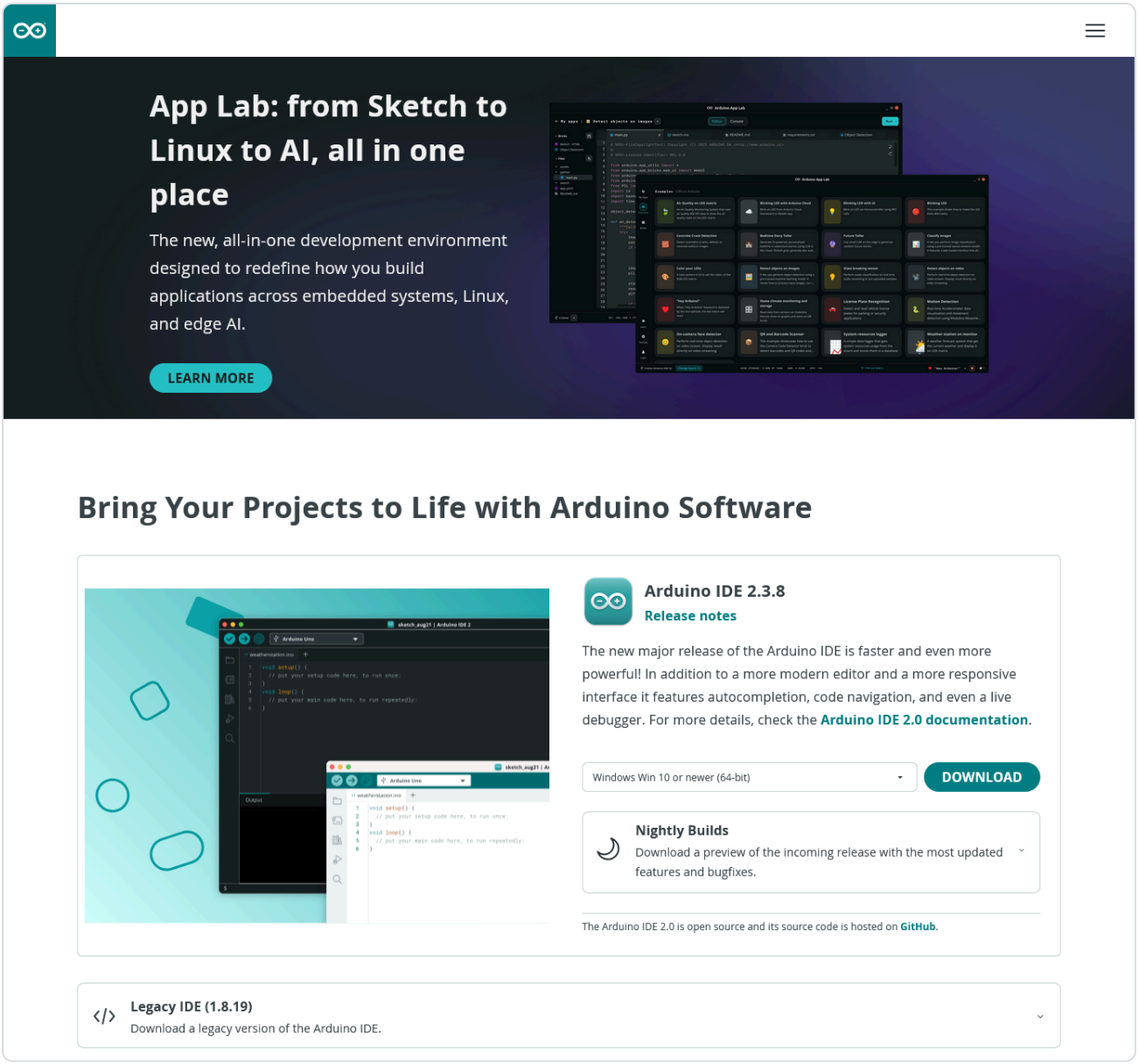
L'IDE (Integrated Development Environment), c'est le logiciel dans lequel vous allez écrire vos programmes Arduino. C'est un éditeur de code spécialisé : vous écrivez votre programme dedans, et il se charge de l'envoyer à la carte Arduino via le câble USB.

Si vous avez utilisé Processing, l'interface va vous sembler familière — c'est normal, l'IDE Arduino s'en est directement inspiré.

Installer l'IDE Arduino

Téléchargement

1. Rendez-vous sur **https://www.arduino.cc/en/software**
2. La page vous propose directement l'**Arduino IDE 2.3.8** (ou version plus récente)
3. Sélectionnez votre système d'exploitation dans le menu déroulant (Windows, Mac ou Linux)
4. Cliquez sur **DOWNLOAD**
5. Lancez le fichier téléchargé et suivez les étapes d'installation



💡 **Windows** : si l'installateur vous demande d'installer des drivers, acceptez. Ce sont les drivers USB nécessaires pour que votre PC communique avec l'Arduino.

⚠️ **Clones chinois (CH340)** : si votre Arduino est un clone avec une puce CH340 (c'est le cas de nos cartes à l'école), le driver s'installe normalement automatiquement sur Windows 10/11. Si la carte n'est pas reconnue, on installera le driver ensemble en classe.

L'interface de l'IDE

Quand vous ouvrez l'IDE Arduino, voici ce que vous voyez :



Zone	Où c'est	À quoi ça sert
Sélecteur de carte	Barre du haut, menu déroulant	Choisir le type d'Arduino et le port USB
Bouton ✓ (Vérifier)	Icône ronde verte, en haut à gauche	Compile le code pour vérifier les erreurs, sans l'envoyer
Bouton → (Téléverser)	Icône flèche verte, à côté du ✓	Compile ET envoie le programme à l'Arduino
Zone de code	Grande zone centrale	C'est là que vous écrivez votre programme
Console	Zone noire en bas	Affiche les messages de compilation et les erreurs
Barre de statut	Tout en bas	Indique la carte connectée, le port COM et la position du curseur

Zone	Où c'est	À quoi ça sert
Explorateur de fichiers	Icône en barre latérale gauche	Naviguer entre les fichiers de votre projet
Moniteur série	Icône loupe en haut à droite	Lire les messages envoyés par l'Arduino (on verra ça plus tard)

Connecter votre Arduino

1. **Branchez l'Arduino** au PC avec le câble USB
2. Dans l'IDE, regardez le **sélecteur de carte** en haut — il devrait automatiquement détecter votre carte
3. Si la carte apparaît (par exemple `Arduino Uno on COM3`), c'est bon, vous êtes connecté !
4. Si rien n'apparaît, cliquez sur le menu déroulant et sélectionnez manuellement :
   - **Type de carte** : `Arduino Uno`
   - **Port** : le port COM qui s'affiche (souvent COM3, COM4 ou COM5 sous Windows)

💡 **Astuce** : si vous ne savez pas quel port correspond à votre Arduino, débranchez-le, regardez la liste des ports, rebranchez-le, et le nouveau port qui apparaît, c'est lui.

Votre premier programme : Blink

Pour vérifier que tout fonctionne, on va charger un programme d'exemple qui fait clignoter la LED intégrée sur l'Arduino.

Ouvrir l'exemple

1. Allez dans **Fichier** → **Exemples** → **01.Basics** → **Blink**
2. Un nouveau fichier s'ouvre avec ce code :

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  // Configure la LED intégrée en sortie
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // Allume la LED
  delay(1000);                      // Attend 1 seconde
  digitalWrite(LED_BUILTIN, LOW);   // Éteint la LED
  delay(1000);                      // Attend 1 seconde
}
```

Téléverser le programme

1. Cliquez sur le **bouton → (Téléverser)** — la flèche verte
2. Attendez que la console affiche `Done uploading.` (Téléversement terminé)
3. Regardez votre Arduino : la petite LED marquée **"L"** près de la pin 13 devrait clignoter !

⚠️ **Si ça ne marche pas** :

Problème	Solution
`Port non disponible`	Vérifiez que le câble USB est bien branché. Essayez un autre câble.
`Carte non reconnue`	Sélectionnez manuellement "Arduino Uno" dans le sélecteur.
`Erreur de téléversement`	Vérifiez que vous avez sélectionné le bon port COM.
La LED ne clignote pas	Attendez 10 secondes — le téléversement peut prendre un moment.

Enregistrer un programme

Pour sauvegarder votre travail :

1. **Fichier → Enregistrer sous...** (ou `Ctrl + Shift + S`)
2. Choisissez un dossier et un nom pour votre projet
3. L'IDE crée automatiquement un dossier avec un fichier `.ino` à l'intérieur

💡 **Convention de nommage** : donnez des noms explicites à vos programmes. Par exemple `Feu_Tricolore`, `Station_Meteo`, `Robot_Obstacle`. Évitez les espaces et les accents dans les noms de fichiers.

⚠️ **Important** : le fichier `.ino` doit toujours être dans un dossier qui porte le même nom. Si votre fichier s'appelle `MonProjet.ino`, il doit être dans un dossier `MonProjet/`. L'IDE gère ça automatiquement quand vous utilisez "Enregistrer sous".

Récapitulatif

Action	Comment faire
Installer l'IDE	arduino.cc/en/software → Télécharger → Installer
Connecter l'Arduino	Brancher USB → Vérifier le sélecteur de carte
Ouvrir un exemple	Fichier → Exemples → 01.Basics → Blink
Vérifier le code	Bouton ✓ (compile sans envoyer)
Téléverser	Bouton → (compile + envoie à l'Arduino)

Action	Comment faire
Enregistrer	Fichier → Enregistrer sous... (`Ctrl + Shift + S`)
Moniteur série	Icône loupe en haut à droite (on verra ça au chapitre suivant)

💡 **Prochain chapitre sur Électro-Lab** : Maintenant que votre IDE est installé et que votre Arduino communique avec le PC, on va pouvoir attaquer les bases de l'électricité et monter nos premiers circuits !

4.2 Potentiomètre et valeurs analogiques

1. Le composant : Le potentiomètre

Le potentiomètre est une résistance dont on peut faire varier la valeur. Contrairement à un bouton poussoir qui n'a que deux états (ouvert ou fermé), le potentiomètre permet de balayer toute une plage de valeurs.

- **Le branchement :** Il possède trois broches. Les deux extrémités se branchent sur l'alimentation (**5V** et **GND**). La broche centrale (le curseur) envoie une tension variable à l'Arduino.
- **Le principe :** Plus on tourne le bouton vers le 5V, plus la tension monte. Plus on tourne vers le GND, plus elle descend vers 0V.

2. La lecture analogique (ADC)

L'Arduino possède des entrées spécifiques nommées **A0 à A5**. Elles sont capables de mesurer cette tension variable. Le processeur traduit cette tension en un nombre compris entre **0 et 1023**.

- **0** correspond à 0 Volt (bouton tourné à fond d'un côté).
- **1023** correspond à 5 Volts (bouton tourné à fond de l'autre côté).

3. La mise à l'échelle (Mapping)

En programmation, on a rarement besoin d'un chiffre entre 0 et 1023. Si on veut piloter un délai, on préfère parler en millisecondes. La fonction `map()` est l'outil indispensable pour convertir ces graduations.

Exemple : `int temps = map(lecture, 0, 1023, 100, 2000);` Ici, on demande à l'ordinateur de transformer l'échelle 0-1023 en une échelle de temps allant de 100ms à 2000ms.

4. Exercice pratique : Réglage d'un clignotement

L'objectif est de construire un programme capable de modifier la vitesse de clignotement d'une LED en temps réel. Nous allons procéder par étapes logiques.

Étape A : Câblage du diviseur de tension

Avant de coder, réalisez le montage suivant. Le potentiomètre va agir comme un capteur de position envoyant une tension variable à l'Arduino.

- **Potentiomètre :** Reliez les bornes extérieures au **5V** et au **GND**. La borne centrale doit être connectée à l'entrée **A0**.
- **LED :** Connectez une LED sur la broche **13** (n'oubliez pas la résistance de protection de 220Ω).

Étape B : Acquisition du signal (La lecture)

Dans la boucle principale `loop()`, la première mission de l'Arduino est d'interroger l'entrée analogique. Le microcontrôleur transforme la tension reçue en un nombre entier.

Logique à implémenter :

```
void loop() {
  // On crée une variable 'lecture' pour stocker le résultat de analogRead()
  int lecture = analogRead(A0);
}
```

Note : À ce stade, `lecture` contient une valeur brute située entre 0 (0V) et 1023 (5V).

Étape C : La fonction de sortie (Le "Blink")

Pour structurer le code, nous créons une fonction isolée qui contient la "recette" du clignotement. Cette fonction doit accepter un paramètre : la durée de l'attente.

```
void maFonctionBlink(int duree) {
  digitalWrite(13, HIGH); // Allumer
  delay(duree);           // Attendre le temps reçu en paramètre
  digitalWrite(13, LOW);  // Éteindre
  delay(duree);           // Attendre à nouveau
}
```

Étape D : Mise à l'échelle (Le Mapping)

Le problème est que `1023` correspond à environ 1 seconde, ce qui est peu réactif. Nous voulons que le potentiomètre gère une plage allant de **50ms** (très rapide) à **1500ms** (très lent).

À ajouter dans le loop() : Utilisez la fonction `map()` pour convertir la variable `lecture`.

```
// Syntaxe : map(valeur, min_entree, max_entree, min_sortie, max_sortie)
int vitesse = map(lecture, 0, 1023, 50, 1500);
```

Étape E : Assemblage final

Il ne reste plus qu'à appeler votre fonction `maFonctionBlink` en lui transmettant votre variable `vitesse`.

Conclusion

Ce module vous a permis de comprendre comment transformer une **grandeur physique continue** (la rotation d'un axe) en une **donnée numérique** exploitable par un programme.

La maîtrise de la lecture analogique est fondamentale en électronique : c'est elle qui permet d'utiliser la majorité des capteurs (température, luminosité, pression, etc.). Gardez cependant à l'esprit que l'utilisation de `delay()` est ici une solution de facilité. Dans un contexte industriel ou pour des systèmes complexes nécessitant de la réactivité, vous devrez apprendre à remplacer ces pauses par une gestion du temps non-bloquante via la fonction `millis()`.

4.3 Le capteur de température (DHT11)

1. Le composant : Le capteur DHT11

Le **DHT11** est un capteur numérique qui permet de mesurer deux données : la température et le taux d'humidité de l'air. Contrairement au potentiomètre qui envoie une tension analogique brute, le DHT11 communique des informations déjà traitées sous forme de données numériques.

Le branchement : Il possède 3 broches.

- **GND** : À relier à la masse.
- **VCC (5V)** : À relier à l'alimentation.
- **DATA** : C'est la broche de signal. Nous la brancherons sur la **broche numérique 4** de l'Arduino.

2. Utiliser une Librairie (Bibliothèque)

L'Arduino est une machine capable de beaucoup de choses, mais il ne sait pas nativement "décoder" le signal complexe du DHT11. Pour lui apprendre, on utilise une **librairie**.

> ***C'est quoi une librairie ?***
>
> *Imagine que tu achètes un meuble en kit. La librairie, c'est la notice de montage. C'est un bloc de code écrit par des experts qui contient toutes les fonctions prêtes à l'emploi. On l'importe pour s'éviter de réécrire des centaines de lignes de calculs compliqués.*

Installation :

1. Allez dans le menu `Outils` > `Gérer les bibliothèques`.
2. Recherchez **"DHT sensor library"** (choisissez celle d'Adafruit).
3. Cliquez sur **Installer**.

3. Configuration du code

Avant d'écrire nos instructions, il faut dire à l'Arduino d'aller lire cette "notice". On ajoute une ligne tout en haut de notre fichier, **avant même le setup**.

```
#include "DHT.h" // On importe la bibliothèque pour gérer le capteur

const int pinCapteur = 4;   // On définit le numéro de la broche (fixe)
DHT dht(pinCapteur, DHT11); // On initialise le capteur sur le pin 4 avec le modèle
DHT11

void setup() {
  // Les configurations initiales iront ici
}

void loop() {
  // La boucle du programme principal ira ici
}
```

Pourquoi cette ligne `#include` ? Elle permet de copier-coller virtuellement tout le code de la bibliothèque dans notre programme. Sans elle, l'ordinateur ne comprendrait pas ce que signifie l'objet `dht`.

4. Initialisation et Lecture

Pour que le capteur s'active, on doit le "réveiller" dans le `setup()`. Ensuite, on récupère les données dans des variables de type `float`.

> ***Le type Float :*** *Jusqu'ici, nous utilisions* `int` *pour les nombres entiers (1, 2, 10). Mais la température est précise : il peut faire 22,5°C. Le type* `float` *est indispensable car il permet de stocker ces nombres à virgule.*

```
void setup() {
  dht.begin(); // On lance la communication avec le capteur
}

void loop() {
  // On crée deux variables pour stocker les mesures
  float humiditee = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Note : Le DHT11 est lent, on doit attendre au moins 2 secondes entre chaque
lecture
  delay(2000);
}
```

5. Le Moniteur Série (L'espion de l'Arduino)

L'Arduino n'a pas d'écran pour nous montrer ce qu'il lit. Pour voir les résultats, on utilise le **Moniteur Série** (accessible via la petite loupe en haut à droite de votre logiciel). C'est un canal de discussion entre l'Arduino et votre ordinateur.

- **Configuration** : Dans le `setup()`, on écrit `Serial.begin(9600);`. Cela définit la vitesse de discussion.
- **Affichage** :
  - `Serial.print()` : Affiche du texte ou une valeur sur la même ligne.
  - `Serial.println()` : Affiche et saute une ligne (Ligne Nouvelle).

Exemple d'affichage :

```
void setup() {
    Serial.begin(9600);
    // le reste du setup
    // ...
}

void loop() {
    // La lecture des données du capteur ici
    // ...

    // Ecrire dans le moniteur
    Serial.print("Temp: ");
    Serial.print(temperature);
    Serial.print("°C | Humidite: ");
    Serial.print(humiditee);
    Serial.println("%");
}
```

6. Exercices pratiques

Niveau 1 : Alerte Humidité 💧

Objectif : Construire un circuit avec une LED sur la broche 13. Le programme doit allumer la LED si le taux d'humidité lu par le DHT11 dépasse 50%.

- *Indice :* Utilisez une condition `if (humiditee > 50.0)`.

Niveau 2 : Le Thermostat Manuel 🌡️

Objectif : On veut pouvoir régler la température d'alerte sans changer le code.

1. Branchez un **potentiomètre** sur l'entrée **A0**.
2. Dans le code, lisez la valeur du potentiomètre (0-1023).
3. Utilisez la fonction `map()` pour transformer cette valeur en une échelle de température allant de **0 à 50°C**. Cela devient votre "seuil".
4. Si la température mesurée par le DHT11 est supérieure à ce seuil, la LED s'allume.

CHAPITRE 5

Drones

1. Sinus et cosinus
2. Comment ça vole ?
3. Introduction à Python
4. Piloter un Tello avec Python

Podcast

https://www.rfi.fr/fr/podcasts/géopolitique/20260425-les-drones-ont-ils-révolutionné-la-guerre

Législation

- Drone Législation.pdf

5.1 Sinus et cosinus, à quoi ça sert ?

> 🎯 ***Objectifs de ce chapitre*** *Après cette lecture, tu sauras :*
>
> - *Expliquer en quoi sin et cos sont différents de Pythagore (et pourquoi on a besoin des deux)*
> - *Identifier l'hypoténuse, le côté opposé et le côté adjacent d'un triangle rectangle*
> - *Utiliser sin et cos pour décomposer un mouvement incliné en composantes horizontale et verticale*
> - *Appliquer sin/cos à des problèmes concrets : rampes, échelles, ombres, vitesses, programmation*
> - *Éviter les pièges classiques (mode degrés/radians, signes, identification des côtés)*

Slides du chapitre

- sin-cos-a-quoi-ca-sert.pdf

Le problème qu'on veut résoudre

Avant de plonger dans les formules, posons-nous **la** question. Imagine ces situations :

- Tu marches 100 m sur une route qui monte. **De combien d'altitude as-tu gagné** à l'arrivée ?
- Une échelle de 6 m est posée contre un mur. **À quelle hauteur arrive son sommet** ?

- Un avion vole à 300 km/h en grimpant. **Quelle est sa vraie vitesse de montée verticale** ?
- Le soleil est à 40° au-dessus de l'horizon. **Quelle longueur fait l'ombre d'un poteau** ?
- Tu pousses un chariot dans une côte inclinée. **Quelle partie de ton effort sert à le faire avancer, quelle partie sert à le soutenir** ?

Toutes ces situations ont un point commun : il y a un **mouvement ou une longueur inclinée**, et on veut savoir **ce qu'il vaut horizontalement, et ce qu'il vaut verticalement**.

> 💡 ***C'est exactement ça, le job de sin et cos : décomposer une longueur (ou un mouvement, ou une force) inclinée en deux parties perpendiculaires.***
>
> *Une longueur **diagonale** → ce qui est **horizontal** + ce qui est **vertical**.*

"Mais on a déjà Pythagore, non ?"

Très bonne objection. Pythagore et sin/cos sont **des outils complémentaires**, pas concurrents. Ils répondent à des questions **différentes** :

Ce que tu connais...	Ce que tu cherches...	Outil à utiliser
Les 2 petits côtés (a et b)	L'hypoténuse c	**Pythagore** : $c^2 = a^2 + b^2$
L'hypoténuse + 1 petit côté	L'autre petit côté	**Pythagore**
1 côté + 1 ANGLE	Les autres côtés	**Sin / Cos**
2 côtés	**Un ANGLE**	**Sin / Cos** (inversés)

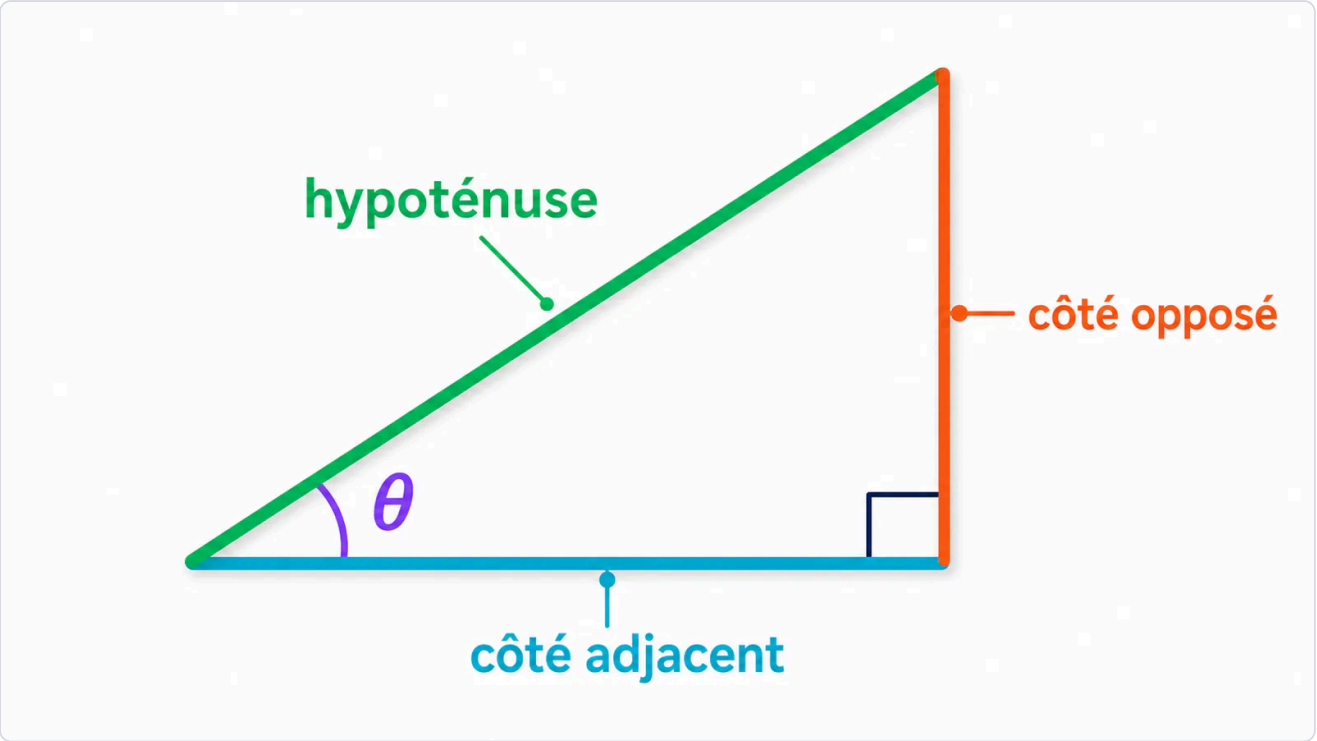
🎯 **La différence fondamentale** :

- **Pythagore** travaille uniquement avec des **longueurs**. Si tu n'as pas d'angle, c'est ton ami.
- **Sin / Cos** travaillent avec des **longueurs ET un angle**. Si tu as un angle dans ton problème, c'est eux qu'il te faut.

Et dans la vraie vie, **on a très souvent un angle** : l'angle d'une rampe, l'inclinaison d'un toit, la position du soleil, l'angle de montée d'un avion, l'inclinaison d'un drone. Pythagore ne sait pas faire grand-chose avec ces problèmes-là.

Le triangle rectangle : un peu de vocabulaire

Sin et cos s'utilisent dans un **triangle rectangle** — un triangle où un des angles fait exactement 90°.



Dans ce triangle, on a trois côtés à connaître :

- **L'hypoténuse** : le plus grand côté, situé **en face de l'angle droit**. C'est lui qui "ferme" le triangle.
- **Le côté adjacent à θ** : c'est le côté qui **touche** l'angle θ (et qui n'est pas l'hypoténuse).
- **Le côté opposé à θ** : c'est le côté **en face** de l'angle θ (qui ne le touche pas).

⚠️ **Piège classique** : les côtés "opposé" et "adjacent" **dépendent de l'angle que tu regardes**. Si tu changes d'angle (les deux angles aigus du triangle), opposé et adjacent s'inversent. L'hypoténuse, elle, ne change jamais — c'est toujours le côté en face de l'angle droit.

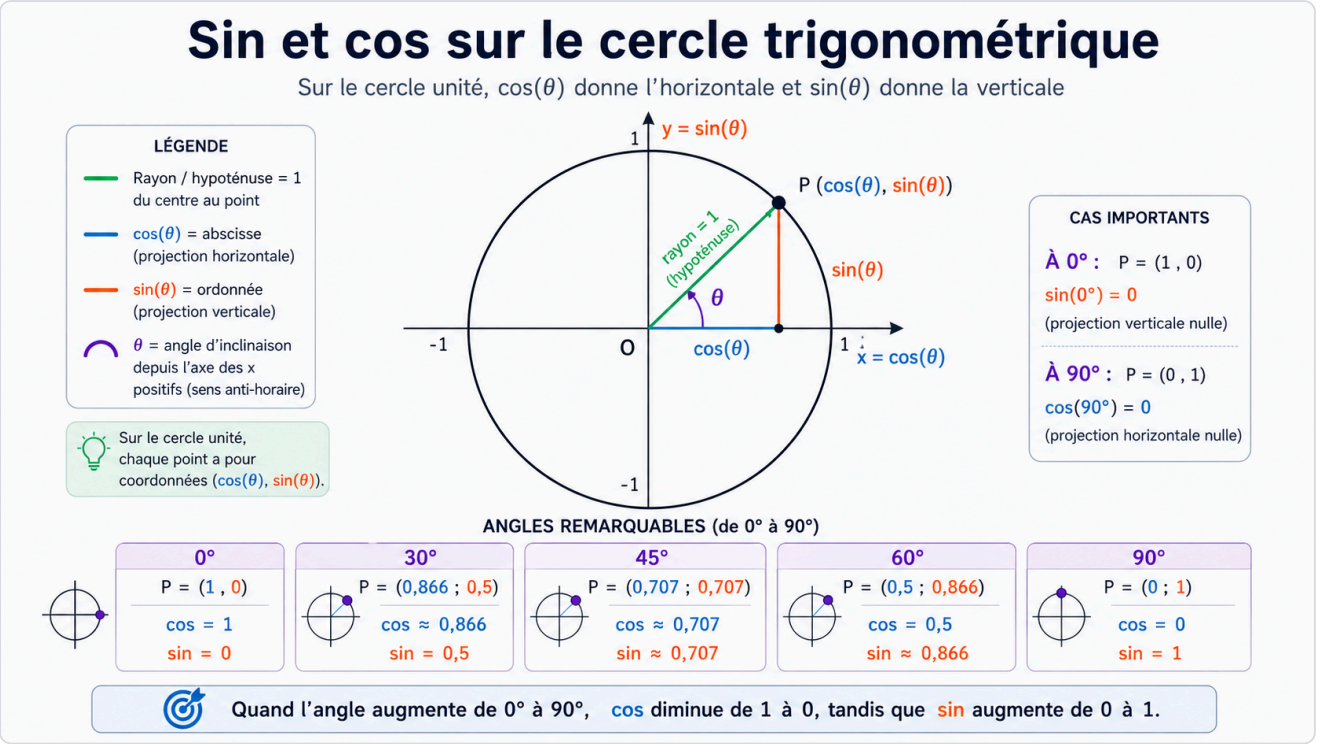
Sin et cos : les définitions

Voici les formules — ce sont juste deux **fractions** très simples :

```
cos(θ) = côté adjacent / hypoténuse
sin(θ) = côté opposé   / hypoténuse
```

C'est tout. Sin et cos sont des **proportions** — des rapports entre deux longueurs. Pour des angles entre 0° et 90°, **les valeurs de sin et cos sont toujours entre 0 et 1**.

L'intuition visuelle (importante !)



Imagine une hypoténuse de **longueur 1** qui pivote autour de l'origine, d'un angle 0° (couchée) jusqu'à 90° (dressée). À chaque instant, regarde sa projection horizontale (cos) et sa projection verticale (sin) :

- **À 0°** : l'hypoténuse est complètement **couchée** à l'horizontale.
  - Sa projection horizontale = 1 (toute la longueur est horizontale) → `cos(0°) = 1`
  - Sa projection verticale = 0 (rien n'est vertical) → `sin(0°) = 0`
- **À 45°** : l'hypoténuse est **inclinée pile au milieu**.
  - Ses deux projections sont égales, à environ 0,71 (~71% de la longueur) chacune.
  - `cos(45°) ≈ 0,71` et `sin(45°) ≈ 0,71`
- **À 90°** : l'hypoténuse est complètement **dressée** à la verticale.
  - Sa projection horizontale = 0 (rien n'est horizontal) → `cos(90°) = 0`
  - Sa projection verticale = 1 (toute la longueur est verticale) → `sin(90°) = 1`

💡 **À retenir** :

- **Le cos te dit à quel point l'hypoténuse "couche" vers l'horizontale.** Plus l'angle est petit, plus cos est grand (proche de 1).
- **Le sin te dit à quel point l'hypoténuse "se dresse" vers la verticale.** Plus l'angle est grand, plus sin est grand (proche de 1).
- **À 45°**, l'inclinaison est partagée 50/50 — c'est pour ça que cos(45°) = sin(45°).

Comment retrouver sin et cos en pratique ?

Trois méthodes selon ton outil :

Méthode 1 — La calculatrice

Toutes les calculatrices (téléphone inclus) ont des touches `sin` et `cos`. ⚠️ **LE piège classique** : il faut être en mode **degrés (DEG)** et non en radians (RAD).

Si tu calcules `sin(30)` en mode RAD, tu obtiens -0,988. En DEG, tu obtiens 0,5. **Vérifie toujours le petit indicateur "DEG" en haut de l'écran avant de calculer.**

Méthode 2 — Les valeurs remarquables à mémoriser

Pour les angles "ronds" (0°, 30°, 45°, 60°, 90°), les valeurs sont fixes et utiles à connaître par cœur :

Angle	cos(θ)	sin(θ)
0°	1	0
30°	≈ 0,87	0,5
45°	≈ 0,71	≈ 0,71
60°	0,5	≈ 0,87
90°	0	1

> 💡 ***Observe la symétrie*** : *`cos(30°) = sin(60°)`, et `cos(60°) = sin(30°)`. Ce n'est pas un hasard — c'est la même propriété du triangle vue depuis ses deux angles aigus complémentaires.*

Méthode 3 — En programmation

En Python : `math.cos(angle)` et `math.sin(angle)`. ⚠️ **Python prend des radians, pas des degrés.** Pour convertir :

```python
import math

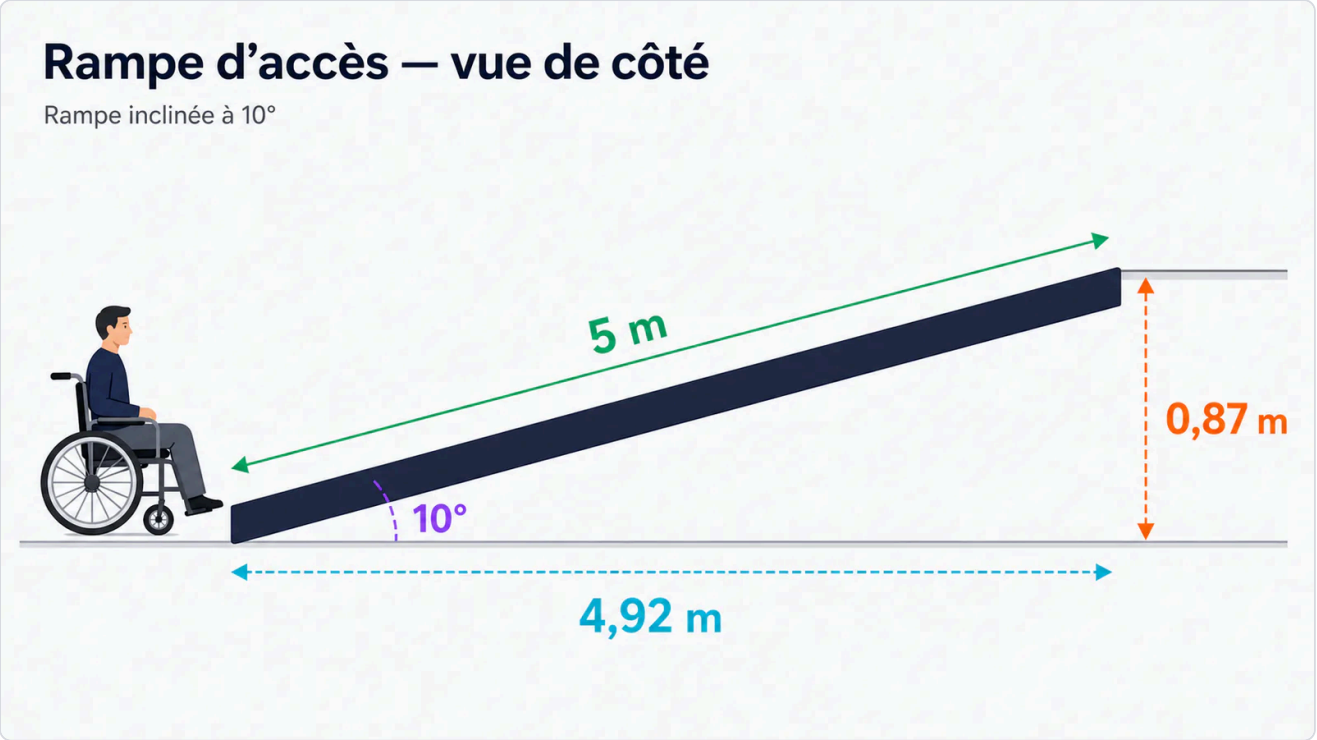
angle_deg = 30
angle_rad = math.radians(angle_deg)   # conversion deg → rad
cos_30 = math.cos(angle_rad)           # ≈ 0,87
sin_30 = math.sin(angle_rad)           # = 0,5
```

En Processing (que tu as utilisé) : pareil, le défaut est en radians. Tu peux changer ce comportement avec `angleMode(DEGREES)` au début de ton programme.

8 exemples concrets — du quotidien au drone

C'est maintenant que ça devient utile. Regarde ces situations qui sont **toutes des décompositions** d'un mouvement ou d'une longueur inclinée.

Exemple 1 — La rampe d'accès handicapé



Une rampe d'accès fait **5 m de long** et est inclinée à **10°** par rapport au sol.

Question 1 : de combien la rampe monte-t-elle au-dessus du sol ?

C'est le côté **opposé** à l'angle de 10° (la hauteur "vers le haut"). On utilise `sin` :

```
hauteur = 5 × sin(10°) = 5 × 0,174 ≈ 0,87 m
```

La rampe monte d'environ **87 cm**.

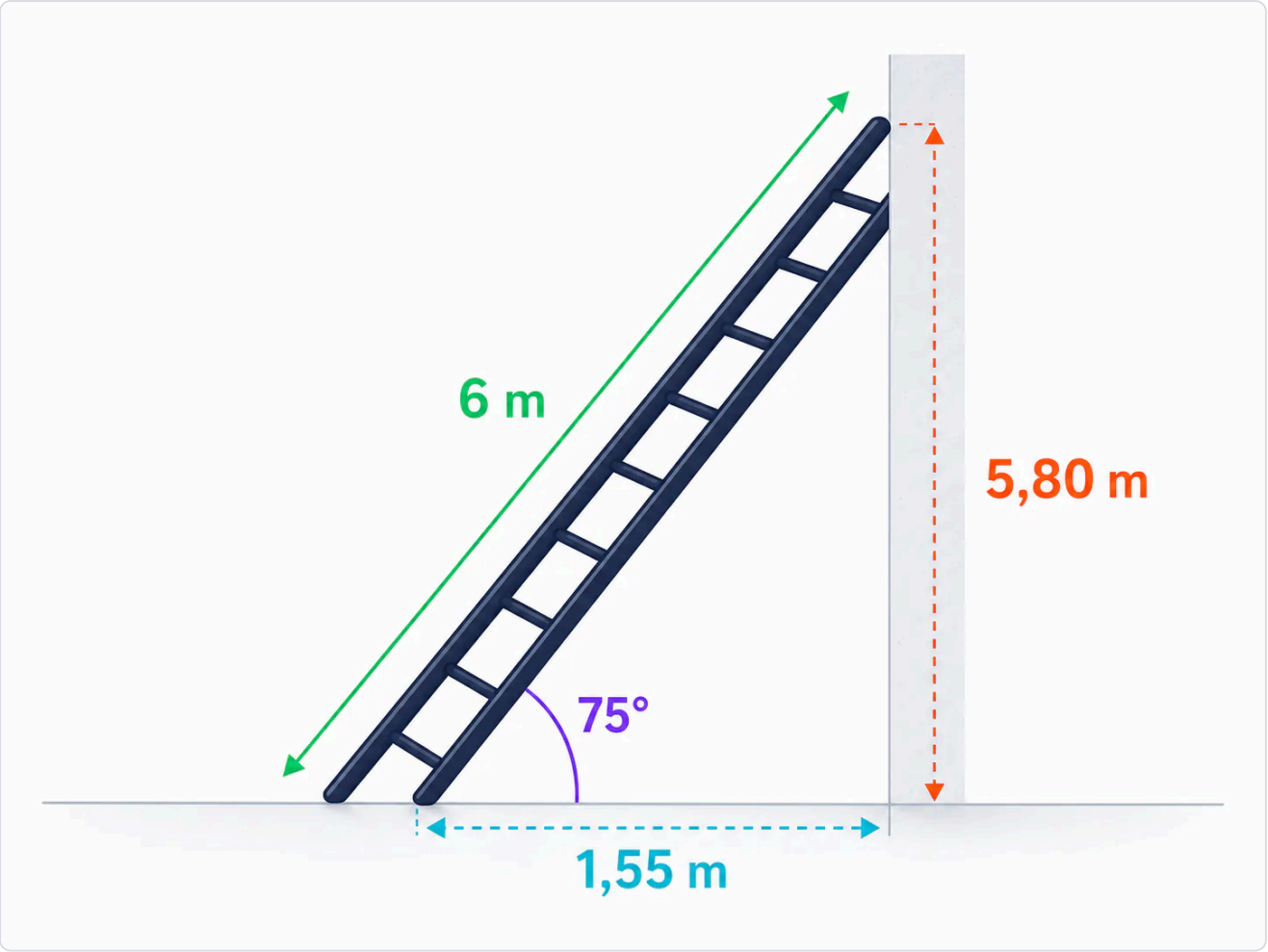
Question 2 : sur quelle distance horizontale s'étend-elle ?

C'est le côté **adjacent**. On utilise `cos` :

```
distance horizontale = 5 × cos(10°) = 5 × 0,985 ≈ 4,92 m
```

La rampe s'étend de **4,92 m** au sol.

Exemple 2 — L'échelle contre le mur



Une échelle de **6 m** est appuyée contre un mur. L'angle entre l'échelle et le sol est de **75°**.

À quelle hauteur arrive le sommet ? (côté opposé à 75°)

```
hauteur = 6 × sin(75°) = 6 × 0,966 ≈ 5,80 m
```

À quelle distance du mur est son pied ? (côté adjacent)

```
distance = 6 × cos(75°) = 6 × 0,259 ≈ 1,55 m
```

> 💡 ***Vérification*** *: avec sin/cos, on peut vérifier que* `sin²(75°) + cos²(75°) = 1`*. Ce qui signifie que* `hauteur² + distance² = échelle²` *(Pythagore !) :* `5,80² + 1,55² = 33,64 + 2,40 = 36,04 ≈ 6²`*.* ✅ *Les deux outils sont cohérents.*

Exemple 3 — La randonnée en montagne

Tu marches **1 000 m** sur un sentier de montagne incliné à **15°**.

Combien d'altitude as-tu gagnée ?

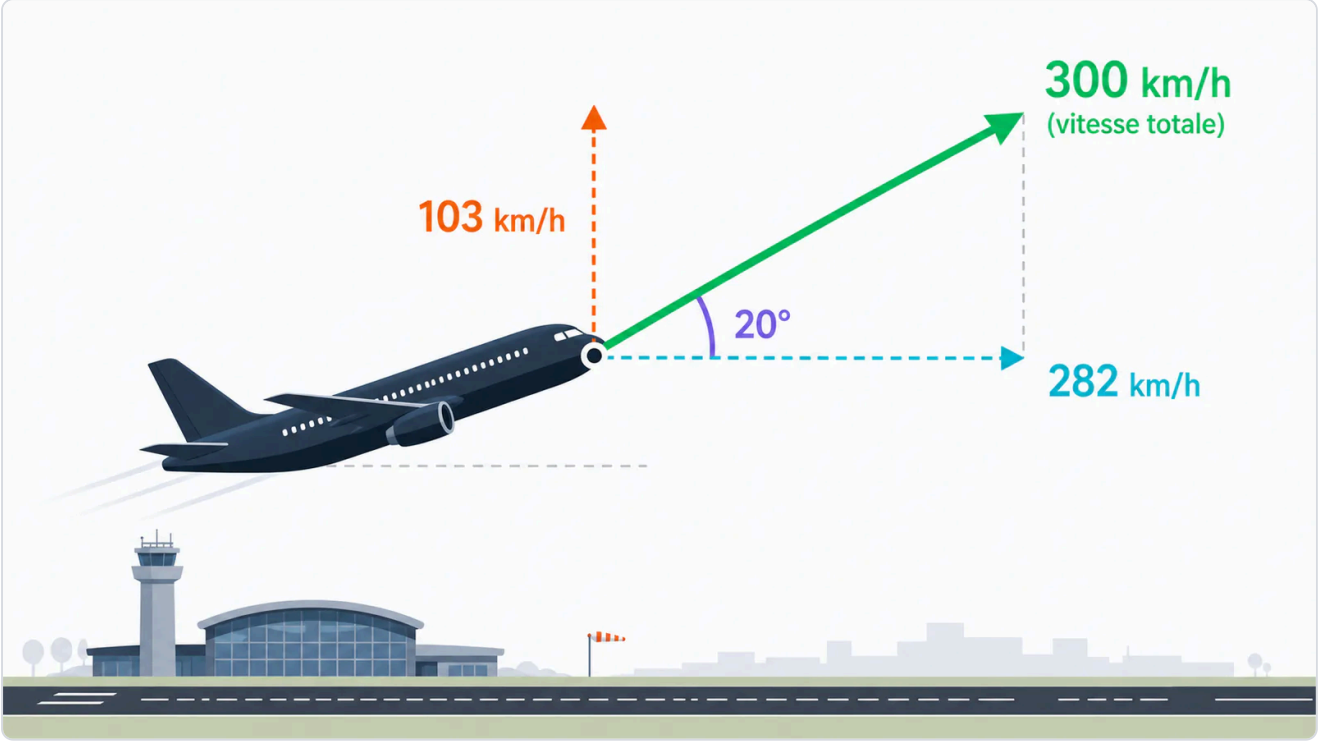
```
gain altitude = 1000 × sin(15°) ≈ 1000 × 0,259 ≈ 259 m
```

Quelle distance as-tu parcourue "à plat" (vue sur la carte) ?

```
distance horizontale = 1000 × cos(15°) ≈ 1000 × 0,966 ≈ 966 m
```

> 💡 ***C'est pour ça qu'une carte topographique "ment" sur les distances*** *: elle montre la distance horizontale (966 m), pas celle que tu as réellement marchée à pied (1000 m). Sur des pentes raides, la différence devient énorme.*

Exemple 4 — L'avion qui décolle



Un avion vole à **300 km/h** en grimpant avec un angle de **20°** par rapport à l'horizontale.

Sa vraie vitesse verticale de montée ?

```
vitesse verticale = 300 × sin(20°) ≈ 300 × 0,342 ≈ 103 km/h
```

Sa vitesse horizontale (sur le sol, comme vue d'un satellite) ?

```
vitesse horizontale = 300 × cos(20°) ≈ 300 × 0,940 ≈ 282 km/h
```

> 💡 *Ici on a décomposé non pas une **longueur** mais une **vitesse**. Le principe est exactement le même : un vecteur (la vitesse de l'avion) incliné à 20°, décomposé en deux composantes perpendiculaires.*

Exemple 5 — Le toit incliné

Tu fais des travaux. Un toit a une **inclinaison de 35°** par rapport à l'horizontale. Le toit fait **8 m de long** mesuré sur la pente.

Quelle est la hauteur du toit (de la gouttière au faîtage) ?

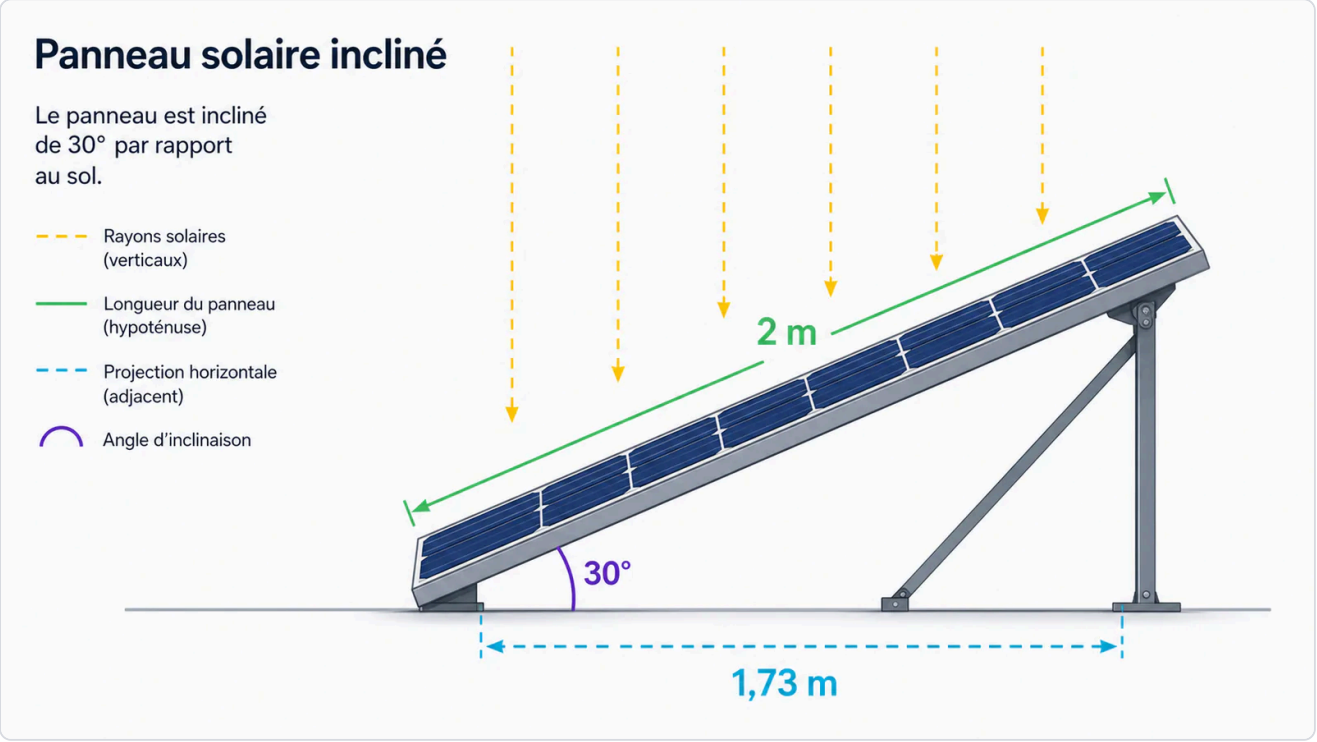
```
hauteur = 8 × sin(35°) ≈ 8 × 0,574 ≈ 4,59 m
```

Quelle est la profondeur du toit (vue de dessus) ?

```
profondeur = 8 × cos(35°) ≈ 8 × 0,819 ≈ 6,55 m
```

C'est exactement ce qu'on calcule pour estimer la surface réelle d'un toit à couvrir d'ardoises, par exemple.

Exemple 6 — Le panneau solaire



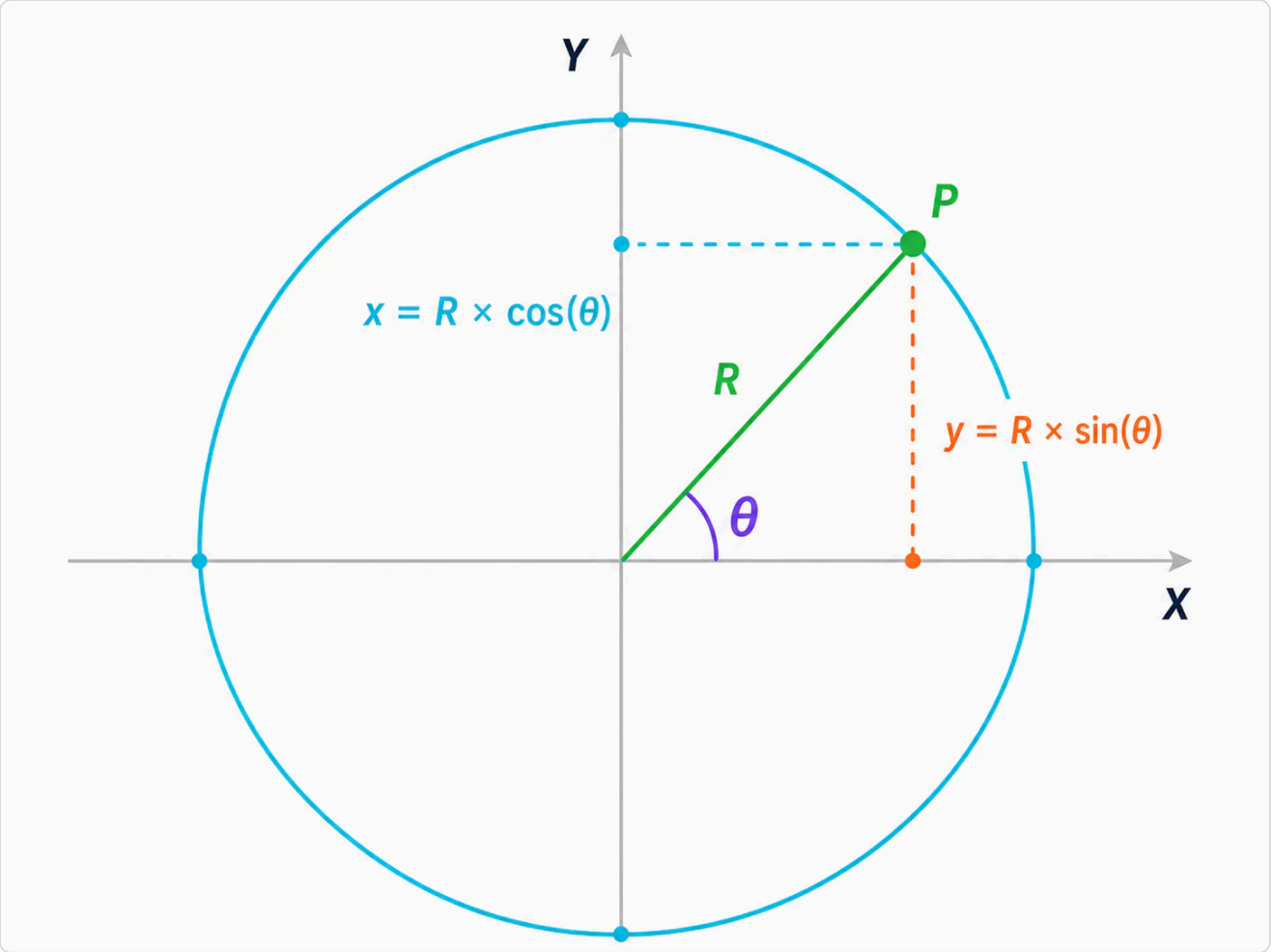
Un panneau solaire de **2 m de large** est incliné à **30°** par rapport au sol pour mieux capter le soleil. Vu de dessus (par le soleil au zénith) :

Quelle "ombre projetée" au sol ? (largeur perçue par le soleil)

```
largeur perçue = 2 × cos(30°) ≈ 2 × 0,866 ≈ 1,73 m
```

Le panneau de 2 m capte donc environ 1,73 m de lumière solaire au lieu de 2 m s'il était à plat... mais quand le soleil est bas (matin/soir), c'est l'inverse qui devient avantageux. **C'est pourquoi l'angle d'inclinaison des panneaux solaires se calcule en fonction de la latitude.**

Exemple 7 — Le code Processing pour dessiner un cercle



Tu as déjà fait ça en cours de programmation. Pour placer un point à un angle θ sur un cercle de rayon **R** centré à l'origine :

```
x = R * cos(θ)
y = R * sin(θ)
```

Pourquoi ça marche ? Le point sur le cercle est à une **distance R du centre** (sur la diagonale). En décomposant cette distance :

- **horizontalement** : `R × cos(θ)`
- **verticalement** : `R × sin(θ)`

C'est exactement la même décomposition que pour la rampe, l'échelle ou l'avion ! Un "rayon" (de longueur R) incliné à un angle θ → composante horizontale + composante verticale.

> 💡 ***C'est pour ça que sin/cos sont partout en programmation graphique*** *: tout ce qui tourne (planètes, pendules, aiguilles d'horloge), tout ce qui oscille (ondes), tout ce qui ondule, repose là-dessus. Pareil pour les animations, les transformations de jeux vidéo, etc.*

Exemple 8 — La poussée du drone Tello (le retour) 🚁

Et voilà, on retombe sur le drone. Un Tello incliné à **20°** vers l'avant qui génère une poussée totale **T** :

- **Composante verticale** (qui soutient le poids) : `T × cos(20°) ≈ T × 0,940`
- **Composante horizontale** (qui fait avancer) : `T × sin(20°) ≈ T × 0,342`

C'est **exactement le même outil** qu'on a utilisé pour tous les exemples précédents. Une seule quantité (la poussée T) inclinée → décomposée en deux composantes perpendiculaires.

La même formule décompose :

- Une longueur (rampe, échelle)
- Une distance parcourue (randonnée, toit)
- Une vitesse (avion)
- Une largeur (panneau solaire)
- Un rayon (programmation graphique)
- Une force (poussée du drone)

> 🎯 ***Le concept est universel*** *: dès qu'une quantité physique est* ***inclinée****, sin/cos te permettent de la séparer en deux composantes perpendiculaires.*

Les 5 pièges classiques

Piège 1 — Mode degrés vs radians

Sur ta calculatrice ou en programmation, vérifie toujours l'unité. Une erreur de mode = résultat catastrophique.

Piège 2 — Le bon côté

"Opposé" et "adjacent" dépendent de **l'angle que tu regardes**. Avant chaque calcul, identifie clairement quel est ton angle, et ses côtés opposé/adjacent.

Piège 3 — Les valeurs hors limites

Pour des angles aigus (0° à 90°), sin et cos sont **toujours entre 0 et 1**. Si ta calculatrice te donne 1,5 ou -0,7, t'as fait une boulette quelque part.

Piège 4 — cos(0°) = 1, pas 0

C'est contre-intuitif au début, mais logique : à 0°, l'hypoténuse est complètement couchée → tout est horizontal → `cos = 1`.

Piège 5 — Les unités du résultat

Si tu calcules `100 m × sin(30°)`, le résultat est en **mètres**. Si tu calcules `200 km/h × sin(20°)`, c'est en **km/h**. Sin et cos eux-mêmes n'ont pas d'unité (ce sont des proportions).

Bonus — La tangente (tan)

Il existe une troisième fonction trigonométrique, la **tangente** :

```
tan(θ) = sin(θ) / cos(θ) = côté opposé / côté adjacent
```

À quoi sert `tan` ? À calculer une **pente ou un angle sans connaître l'hypoténuse**.

Exemple : tu vois un mur de 3 m et son ombre fait 1 m au sol. Quel angle fait le soleil par rapport à l'horizon ?

```
tan(θ) = 3 / 1 = 3
θ = arctan(3) ≈ 71,6°
```

On utilise `tan` constamment dans :

- Les **pentes routières** (% de pente = tan × 100). Une pente de 10% = `tan(θ) = 0,10`, donc θ ≈ 5,7°
- La **photographie** (angle de champ)
- La **navigation** (gisements)

Synthèse — Ce qu'il faut retenir

1. **Sin et cos décomposent une quantité (longueur, vitesse, force) inclinée** en deux composantes perpendiculaires (horizontale + verticale).
2. **Tu en as besoin** quand tu connais **1 côté + 1 angle** d'un triangle rectangle et que tu cherches les autres côtés. Pythagore, lui, ne marche qu'avec des longueurs entre elles.

3. **Les formules** : `cos(θ) = adjacent/hypoténuse`, `sin(θ) = opposé/hypoténuse`.
4. **Sin et cos sont des proportions** entre 0 et 1 (pour des angles aigus). Ce sont des nombres sans unité.
5. **En pratique** : `longueur × cos(θ)` donne la composante adjacent (souvent horizontale), `longueur × sin(θ)` donne la composante opposée (souvent verticale).
6. **C'est partout** : rampes, échelles, toits, panneaux solaires, avions, randonnées, programmation graphique, drones. Une fois compris, tu réutiliseras ces formules toute ta vie.

Pourquoi ça vaut le coup

Sin et cos ne sont pas juste "des maths du lycée". Ce sont les **outils fondamentaux** de :

- La **physique** (toute force ou vitesse inclinée se décompose en sin/cos)
- La **programmation graphique** (rotations, cercles, animations, jeux)
- L'**ingénierie civile** (rampes, structures, mécanique)
- La **navigation** (cartes, GPS, ombres, soleil, gisements)
- Le **pilotage de drones et d'avions** ✈️

> 🧠 ***Une fois que tu as ancré l'idée que sin/cos = décomposition***, *tu réutiliseras ces formules sans même y penser. C'est comme apprendre à lire : ça paraît compliqué au début, et ensuite ça devient automatique.*

Et la prochaine fois que tu verras un drone s'incliner pour avancer, tu sauras précisément **pourquoi** et **comment** ça marche. 🚁

5.2 Comment ça vole ?

> 🎯 ***Objectifs de ce chapitre*** *À la fin de cette séance, tu sauras :*
>
> - *Expliquer pourquoi un objet de 80 grammes peut rester suspendu en l'air*
> - *Identifier les principaux composants d'un quadrirotor et leur rôle*
> - *Décrire les 4 commandes de pilotage et comment elles modifient les vitesses des hélices*
> - *Reconnaître les capteurs embarqués et comprendre leur fonctionnement*
> - *Faire le lien entre la boucle de contrôle d'un drone et le `loop()` d'un programme Arduino*
> - *Effectuer un premier vol manuel avec un Tello en toute sécurité*

Slides du chapitre

- drones-s01-comment-ca-vole.pdf

Le mystère du vol

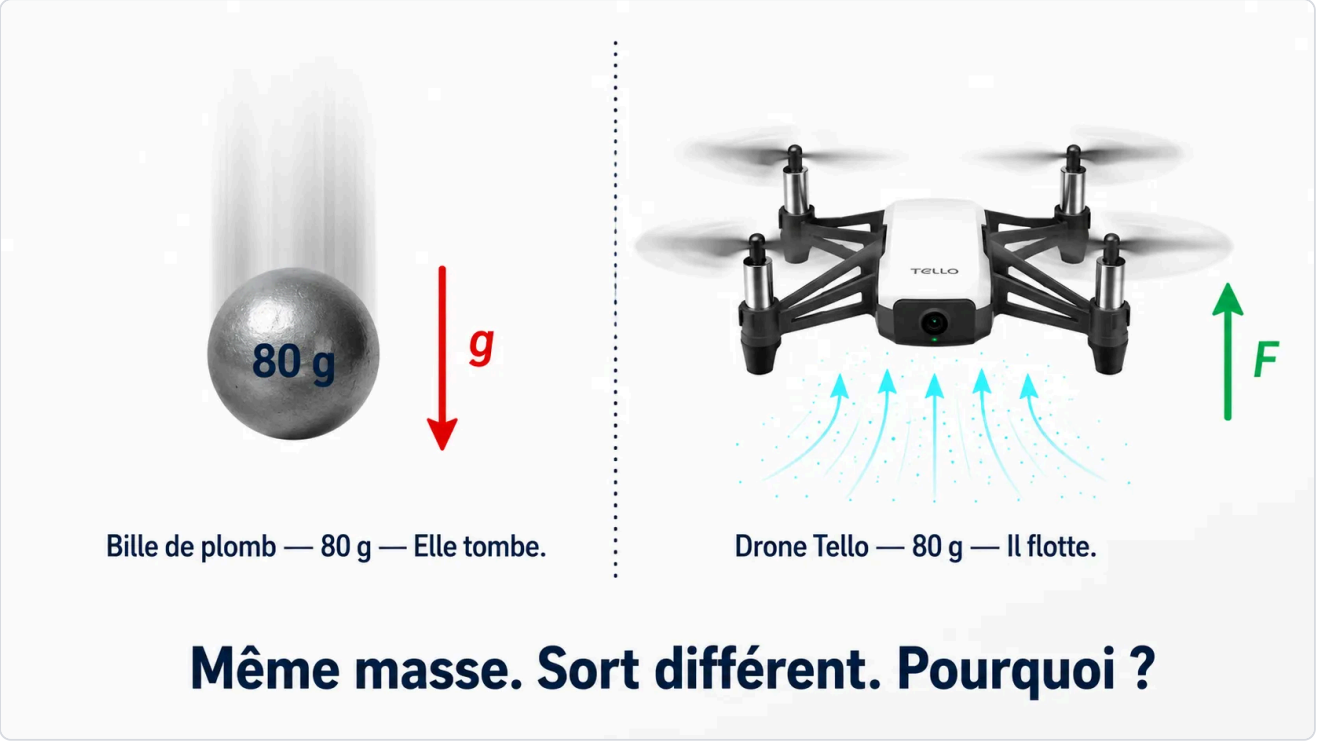
Imagine deux objets identiques par leur poids : une bille de plomb de 80 grammes, et un drone Tello de 80 grammes. Lance-les en l'air. La bille retombe. Le Tello, lui, **flotte**.

Comment est-ce possible ? Les deux objets subissent **la même gravité**, et pourtant l'un tombe et l'autre reste en l'air. La différence ne vient ni de la matière, ni de la forme, ni du hasard. Elle vient de **l'énergie embarquée**.

Le Tello possède une batterie qui alimente 4 moteurs. Ces moteurs entraînent 4 hélices qui projettent de l'air vers le bas, en permanence, à très grande vitesse. Et c'est là qu'intervient une loi fondamentale de la physique : **la troisième loi de Newton**.

> 📜 ***Troisième loi de Newton*** *: Pour toute action, il existe une réaction égale et opposée.*

Concrètement : si l'hélice pousse l'air vers le bas avec une certaine force, l'air pousse l'hélice (et donc le drone) vers le haut avec **exactement la même force**. C'est ce qui se passe quand tu nages : tu pousses l'eau vers l'arrière, l'eau te pousse vers l'avant. Quand tu sautes, tu pousses la terre vers le bas (très légèrement), et la terre te pousse vers le haut. Le drone fait la même chose, mais avec de l'air. **Tout le temps. Avec quatre hélices à la fois.**



Combien faut-il pousser pour flotter ?

Pour rester en stationnaire, les hélices doivent générer **exactement** la force inverse du poids. Tu te souviens de la formule du poids ?

```
F = m × g

où :
  F = la force (en newtons, N)
  m = la masse (en kilogrammes, kg)
  g = l'accélération de la pesanteur, environ 9,81 m/s² sur Terre
```

Pour notre Tello de 80 grammes :

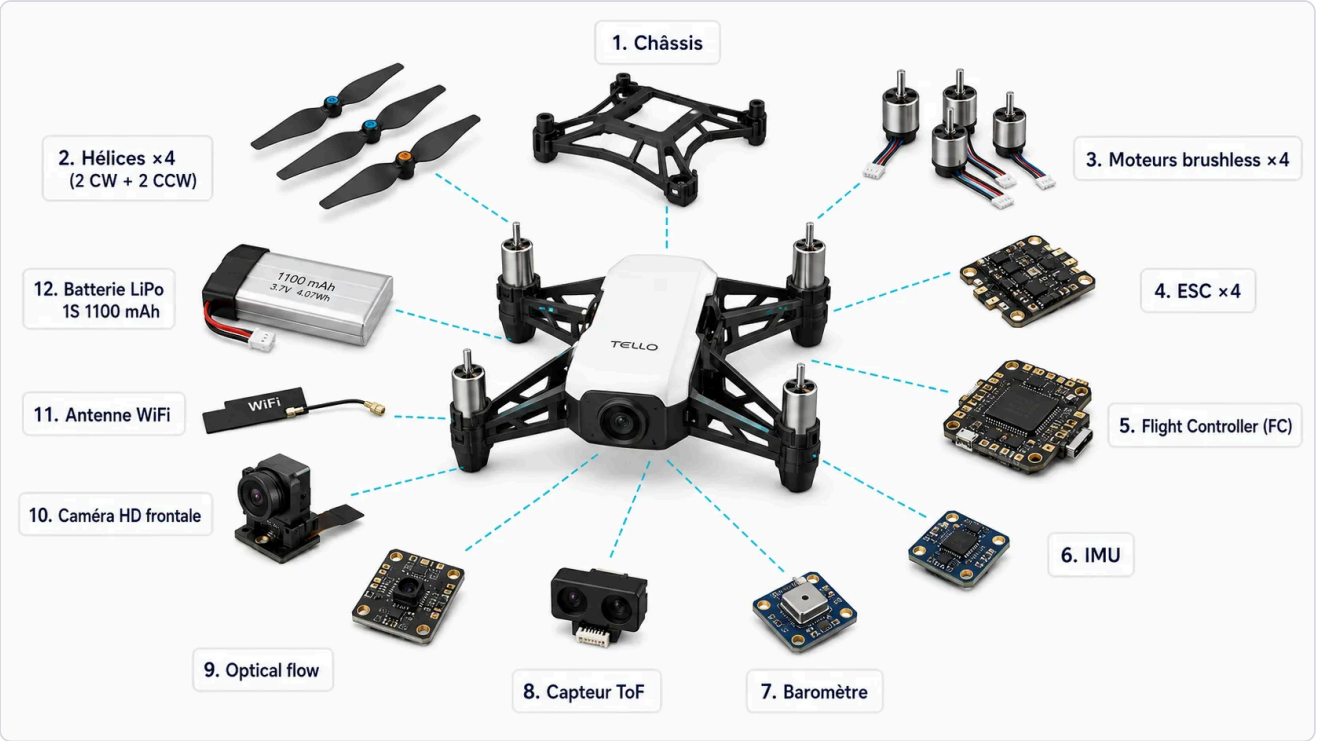
```
m = 80 g = 0,080 kg
F = 0,080 × 9,81
F ≈ 0,78 newtons
```

Donc les 4 hélices du Tello doivent générer **environ 0,78 newtons de poussée** pour le maintenir en l'air. Pour avoir une idée concrète : 1 newton, c'est à peu près le poids d'une pomme moyenne. Le Tello en stationnaire, c'est donc l'équivalent du poids d'une grosse fraise qui se tient en l'air toute seule. Si la poussée totale dépasse 0,78 N, le drone monte. Si elle est inférieure, il descend.

💡 **Pas de magie, juste de la physique** : un drone n'est pas plus mystérieux qu'un avion ou un oiseau. Il échange constamment de l'énergie (batterie → moteurs → air → poussée) pour rester en l'air.

Anatomie d'un quadrirotor

Avant d'aller plus loin, regardons ce qu'il y a sous le capot. Un drone comme le Tello, c'est essentiellement **un Arduino dopé aux stéroïdes**, branché à 4 moteurs très rapides et bourré de capteurs.



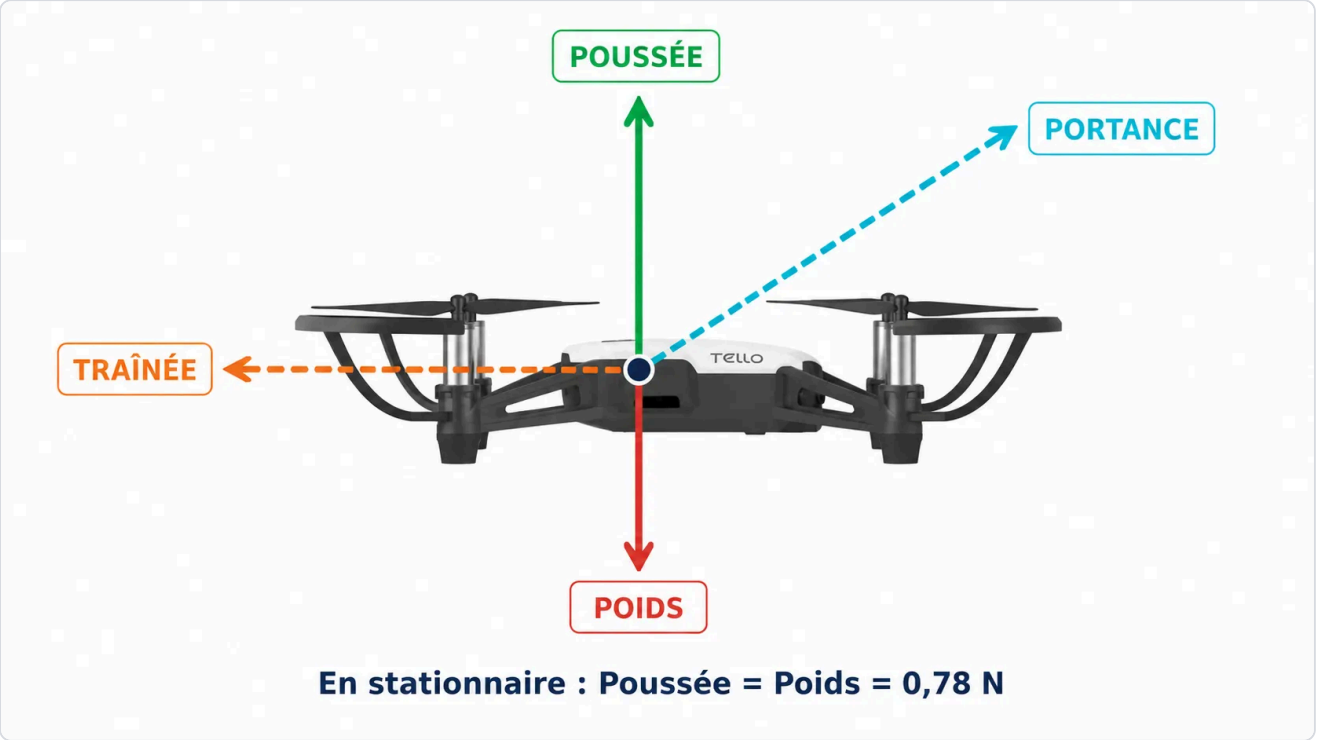
Composant	Rôle	Tu connais déjà ?
Châssis	Structure légère qui tient tout ensemble	Comme le châssis du robot 2WD Arduino
4 moteurs brushless	Font tourner les hélices très vite (jusqu'à 30 000 tours/min)	Cousins des moteurs DC, mais sans charbons
4 ESC (Electronic Speed Controller)	Convertissent un signal PWM en vitesse moteur	Équivalent du **L298N**, mais pour brushless
4 hélices	Génèrent la poussée en projetant l'air vers le bas	Nouveau composant
Flight Controller (FC)	Le "cerveau" qui orchestre tout	Comme un **ESP32 spécialisé**, mais ultra-rapide
IMU (gyro + accéléromètre)	Sait dans quelle position et à quelle vitesse il tourne	Nouveau capteur
Baromètre	Mesure la pression de l'air pour estimer l'altitude	Nouveau capteur
ToF (Time of Flight)	Mesure la distance au sol par laser	Cousin miniature du **HC-SR04**
Optical flow	Caméra sous le drone qui voit le sol défiler	Analogue aux **encodeurs EV3**
Antenne WiFi	Communique avec ton téléphone ou ton script Python	Comme l'**ESP32**

Composant	Rôle	Tu connais déjà ?
Batterie LiPo	Alimente le tout (1100 mAh sur Tello)	Comme les batteries du robot 2WD
Caméra frontale HD	Vidéo pour le pilote ou le programme	Nouveau

💡 **Le truc à retenir** : presque tout ce qui te paraît mystérieux dans un drone, tu en as déjà manipulé une version "Arduino" sur le robot 2WD ou l'EV3. Le drone, c'est juste **plus rapide, plus précis, et en 3D**.

Les 4 forces du vol

Quand un drone vole, **4 forces** agissent sur lui en permanence. Ce sont exactement les mêmes forces qui agissent sur un avion, un parapente ou un oiseau.



Force	Direction	Origine
Poids (P)	Toujours vers le bas	Gravité × masse
Poussée (T pour *thrust*)	Vers le haut (au repos)	Hélices qui projettent l'air vers le bas
Portance (L pour *lift*)	Perpendiculaire au sens de vol	Forme aérodynamique (faible sur les drones)
Traînée (D pour *drag*)	Opposée au sens de déplacement	Frottement de l'air sur le drone

Le cas le plus simple : le stationnaire

Quand le drone est immobile en l'air (mode "hover"), les choses sont simples :

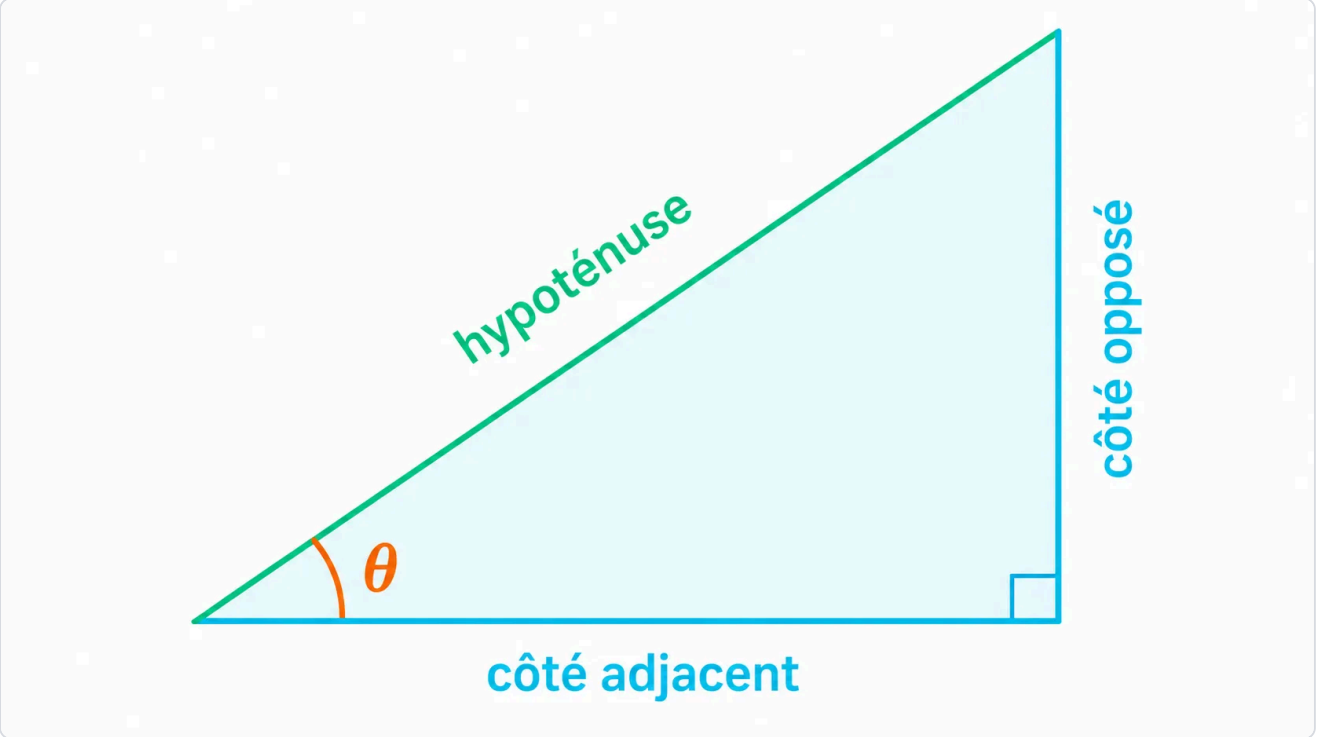
```
Poussée = Poids
0,78 N = 0,78 N
→ Drone immobile en l'air ✓
```

Quand ça se complique : le drone qui se déplace

Si le drone avance, il **s'incline** légèrement vers l'avant. Sa poussée n'est plus verticale : elle est dirigée vers le haut **et** vers l'avant, dans une direction inclinée. Cette poussée inclinée doit toujours soutenir le drone (sinon il tombe) **et** le pousser vers l'avant (sinon il n'avance pas). Pour comprendre comment, on va avoir besoin d'un petit détour mathématique.

Petit détour mathématique : sin et cos

Ne pars pas en courant, je vais le faire vraiment simplement. Le **sinus** et le **cosinus**, ce sont deux outils mathématiques qu'on utilise pour **décomposer un mouvement incliné en deux mouvements perpendiculaires**. Et oui, c'est exactement ce qu'on a besoin pour comprendre le vol incliné du drone.



Le triangle rectangle

Imagine un triangle rectangle. Il a trois côtés :

- L'**hypoténuse** : le côté le plus long (en face de l'angle droit)
- Le côté **adjacent** à un angle θ (le côté qui touche cet angle, à côté de l'angle droit)
- Le côté **opposé** à cet angle (le côté qui lui fait face)

Les fonctions sin et cos te disent **dans quelle proportion** chaque côté représente l'hypoténuse, en fonction de l'angle :

```
cos(θ) = côté adjacent / hypoténuse
sin(θ) = côté opposé / hypoténuse
```

Un exemple concret pour visualiser

Imagine que tu tires une corde de 10 mètres pour la fixer au sommet d'un mât. La corde fait un angle de 30° avec le mât.

```
Hypoténuse (la corde) = 10 m
Angle θ entre la corde et le mât = 30°

Hauteur du mât (côté adjacent à 30°) = 10 × cos(30°) ≈ 10 × 0,866 ≈ 8,66 m
Distance horizontale (côté opposé)   = 10 × sin(30°) ≈ 10 × 0,500 ≈ 5,00 m
```

Donc la corde de 10 mètres, inclinée à 30°, **soutient** la hauteur du mât (8,66 m vers le haut) **et** s'étend (5 m horizontalement). Une seule longueur de corde, deux effets séparés.

Pourquoi c'est utile pour notre drone

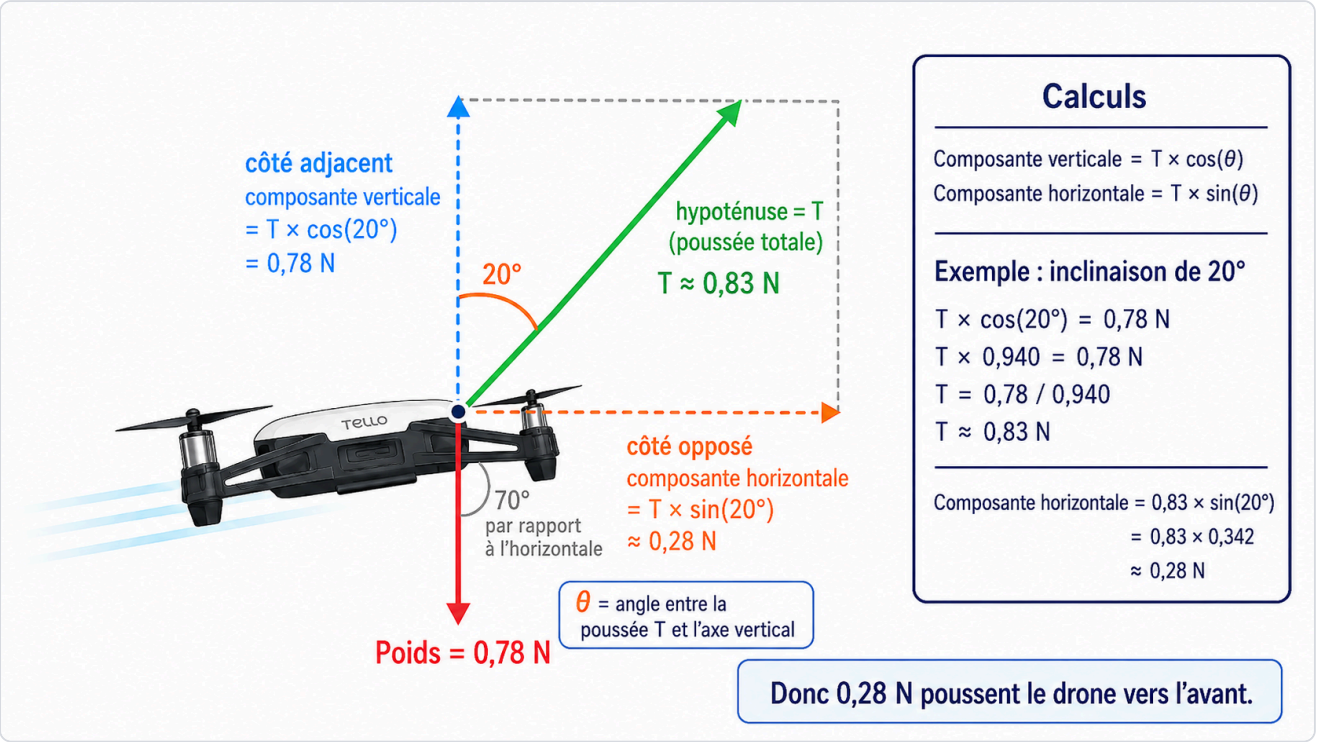
Eh bien : **la poussée d'un drone incliné, c'est exactement la même histoire**. Une seule force (la poussée des hélices), deux effets séparés (soutenir le poids vers le haut + déplacer vers l'avant). On va voir comment juste après.

💡 **Pont avec la programmation** : tu te souviens, en Processing, on a écrit du code pour tracer des cercles et des polygones. Pour calculer la position des sommets d'un hexagone autour d'un centre, on utilisait `x = cos(angle) * rayon` et `y = sin(angle) * rayon`. **C'est exactement les mêmes formules.** La géométrie qu'on a faite en cours de programmation, c'est la physique du drone.

> 💡 ***Pour aller plus loin sur sin et cos*** *: un chapitre dédié "Sinus et cosinus, à quoi ça sert ?" est disponible dans la boîte à outils mathématiques du cours. Il propose 8 exemples concrets (rampe, échelle, randonnée, avion, toit, panneau solaire, code, drone) pour ancrer le concept de décomposition.*

Application au vol incliné

Reprenons notre drone qui avance. Il s'incline vers l'avant d'un angle θ (theta). Sa poussée T se décompose en deux :




```
Composante verticale   = T × cos(θ)    ← cette partie soutient le poids
Composante horizontale = T × sin(θ)    ← cette partie fait avancer le drone
```

Exemple chiffré : drone incliné de 20°

Supposons que le drone Tello s'incline de 20° vers l'avant pour avancer. Pour rester en l'air (ne pas tomber), il faut que sa composante verticale reste égale au poids (0,78 N). On peut calculer la poussée totale nécessaire :

```
T × cos(20°) = 0,78 N
T × 0,940    = 0,78 N
T            = 0,78 / 0,940
T            ≈ 0,83 N
```

Le drone doit donc générer **0,83 N de poussée totale** (au lieu de 0,78 N à l'arrêt). Et combien va vers l'avant ?

```
Composante horizontale = 0,83 × sin(20°)
                       = 0,83 × 0,342
                       ≈ 0,28 N
```

Donc **0,28 newton** poussent le drone vers l'avant. C'est cette force qui le fait accélérer.

💡 **Conclusion importante** : plus le drone s'incline, plus il avance vite. Mais aussi, plus il doit générer de poussée totale pour ne pas perdre d'altitude. À 90° d'inclinaison (impossible en vrai), toute la poussée irait à l'horizontale et le drone tomberait comme une pierre. C'est pour ça que les drones ont une **limite d'inclinaison** dans leur firmware (autour de 30-35° max pour le Tello).

Pourquoi 4 hélices et pas une seule ?

Bonne question. Si une seule hélice peut faire voler un hélicoptère, pourquoi un drone en a-t-il quatre ?

La réponse est encore dans la troisième loi de Newton. Quand un moteur fait tourner une hélice dans un sens, l'hélice exerce un couple sur l'air… et l'air exerce un couple égal et opposé sur le moteur. Concrètement : **si l'hélice tourne dans le sens horaire, le corps du drone aurait tendance à tourner dans le sens antihoraire.**



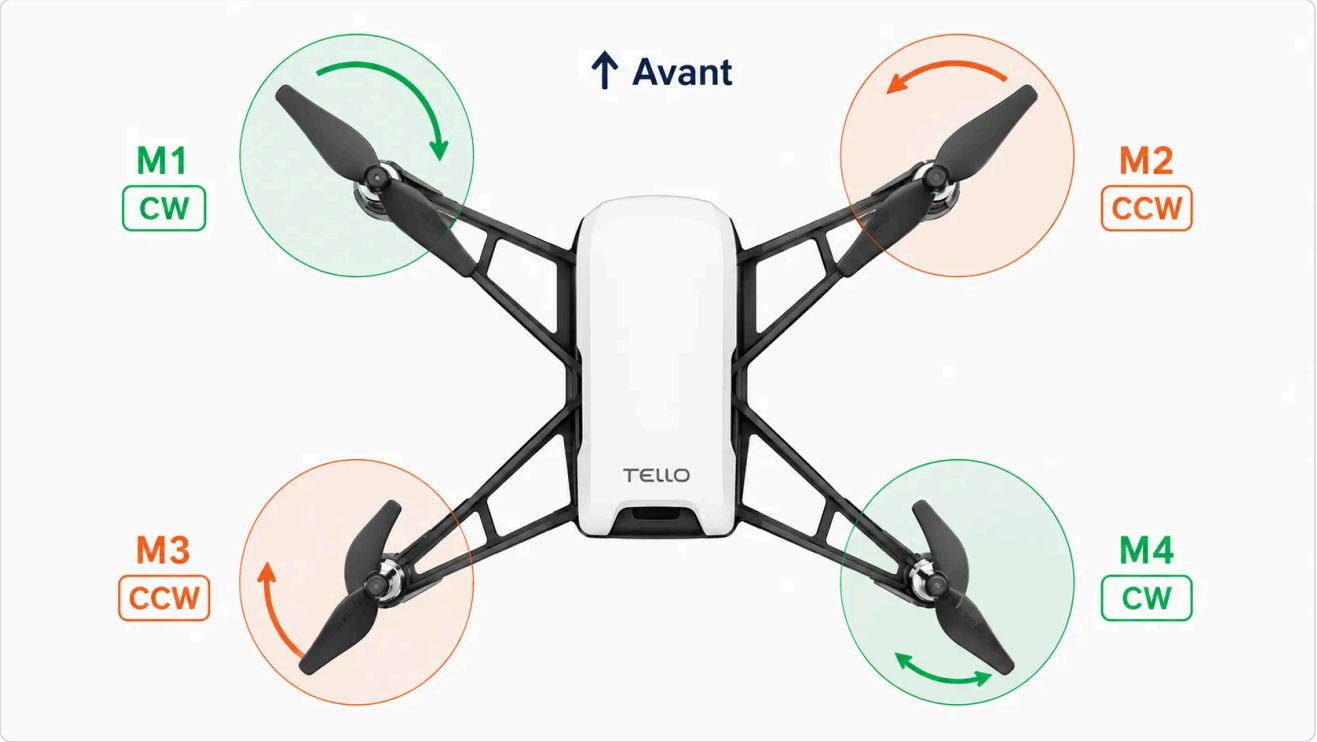
Les hélicoptères résolvent ce problème avec un **petit rotor de queue** qui contre cette rotation parasite (c'est pour ça qu'un hélicoptère a une grande queue et une petite hélice à l'arrière). Élégant, mais mécaniquement complexe.

Les drones, eux, utilisent une astuce plus simple : **2 hélices tournent dans un sens, 2 dans l'autre**. Vue de dessus du Tello :

```
          AVANT
            ↑
M1 ↻          M2 ↺
      \     /
      [TELLO]
      /     \
M3 ↺          M4 ↻
          ARRIÈRE
```

- M1 (avant-gauche) et M4 (arrière-droite) tournent **dans le sens horaire (CW)**
- M2 (avant-droite) et M3 (arrière-gauche) tournent **dans le sens antihoraire (CCW)**

Les couples de réaction s'annulent **deux par deux** : les CW génèrent un couple qui ferait tourner le drone à gauche, les CCW génèrent un couple qui ferait tourner le drone à droite — résultat, le drone reste **stable en lacet** (yaw).



C'est aussi cette configuration intelligente qui permet de **faire tourner volontairement le drone** sur lui-même, comme on va le voir maintenant.

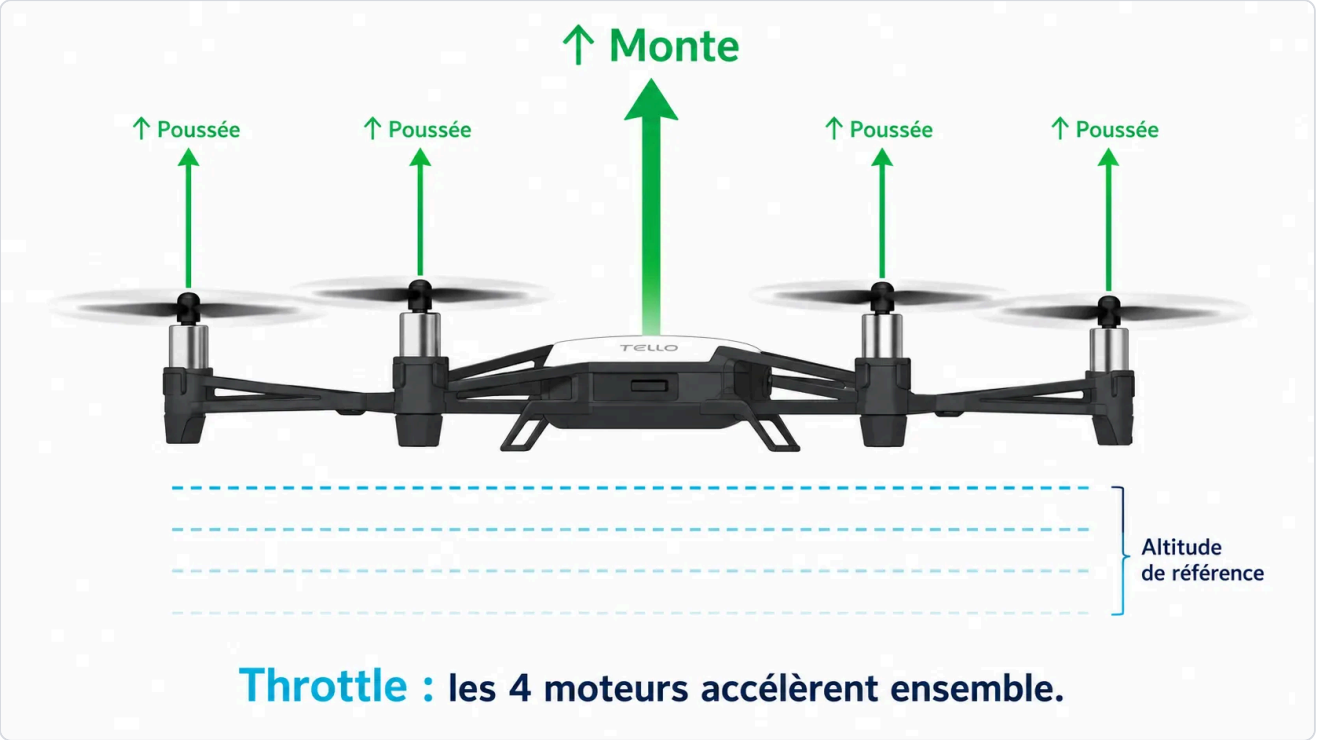
Les 4 commandes de pilotage

Avec 4 hélices et leur configuration CW/CCW, on peut produire exactement **4 commandes de pilotage**. Chacune correspond à une façon différente de moduler les vitesses des 4 moteurs.

Commande	Effet	Combinaison de moteurs
Throttle (gaz)	Monter / descendre	Les 4 moteurs ↑ ensemble
Yaw (lacet)	Pivoter sur soi-même	2 CW ↑ + 2 CCW ↓
Pitch (tangage)	Avancer / reculer	Arrière ↑ + Avant ↓
Roll (roulis)	Translater gauche / droite	Gauche ↑ + Droite ↓

Throttle : monter et descendre

C'est la commande la plus simple. **Les 4 moteurs accélèrent (ou ralentissent) en synchro**, à la même vitesse.

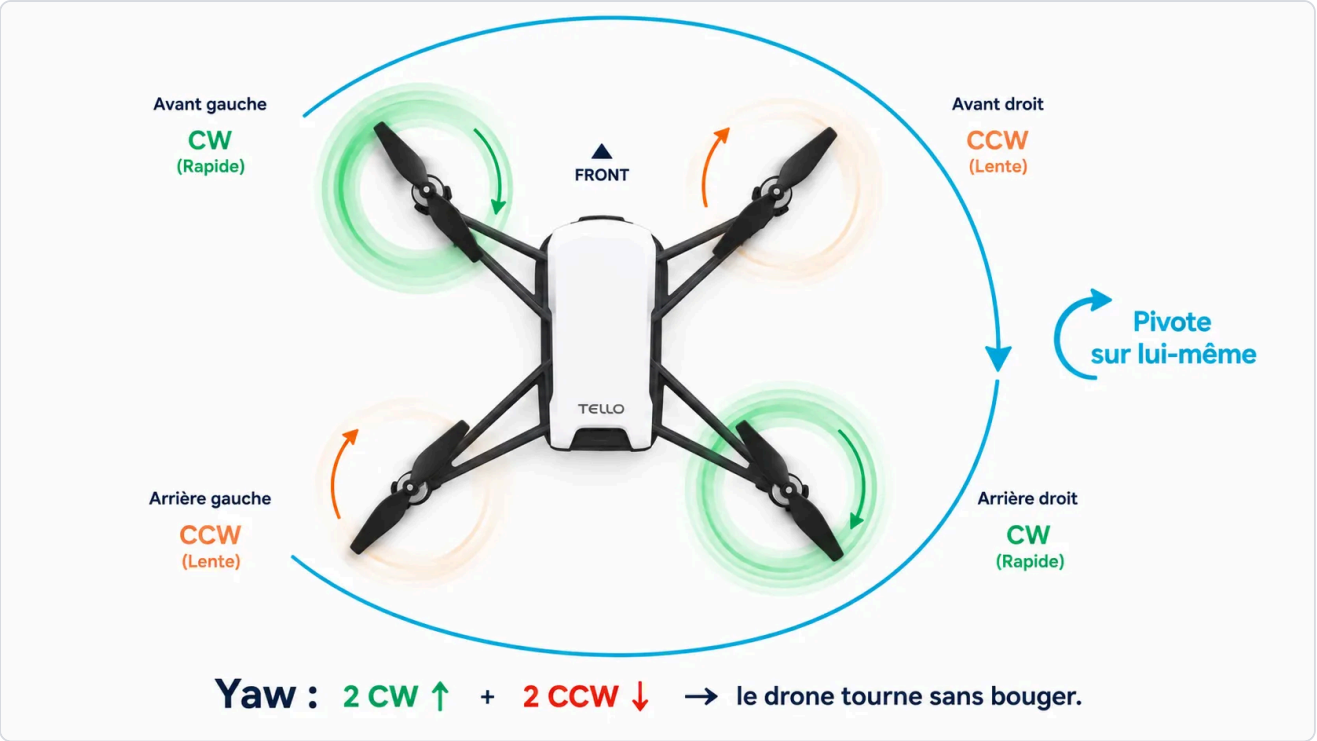


- Si la poussée totale **dépasse** le poids → le drone **monte**
- Si la poussée totale **égale** le poids → le drone **stationne**
- Si la poussée totale **est inférieure** au poids → le drone **descend** (mais lentement, ce n'est PAS la même chose que "couper les moteurs", qui causerait une chute libre)

Exemple concret : sur l'app Tello, le stick gauche poussé vers le haut = throttle augmente = drone monte. Stick gauche tiré vers le bas = throttle diminue = drone descend.

Yaw : pivoter sur soi-même

Le yaw, c'est le mouvement de rotation autour de l'axe vertical, **sans changer de position**. Le drone reste à la même altitude, au même endroit, mais il tourne sur lui-même (comme un toupie).



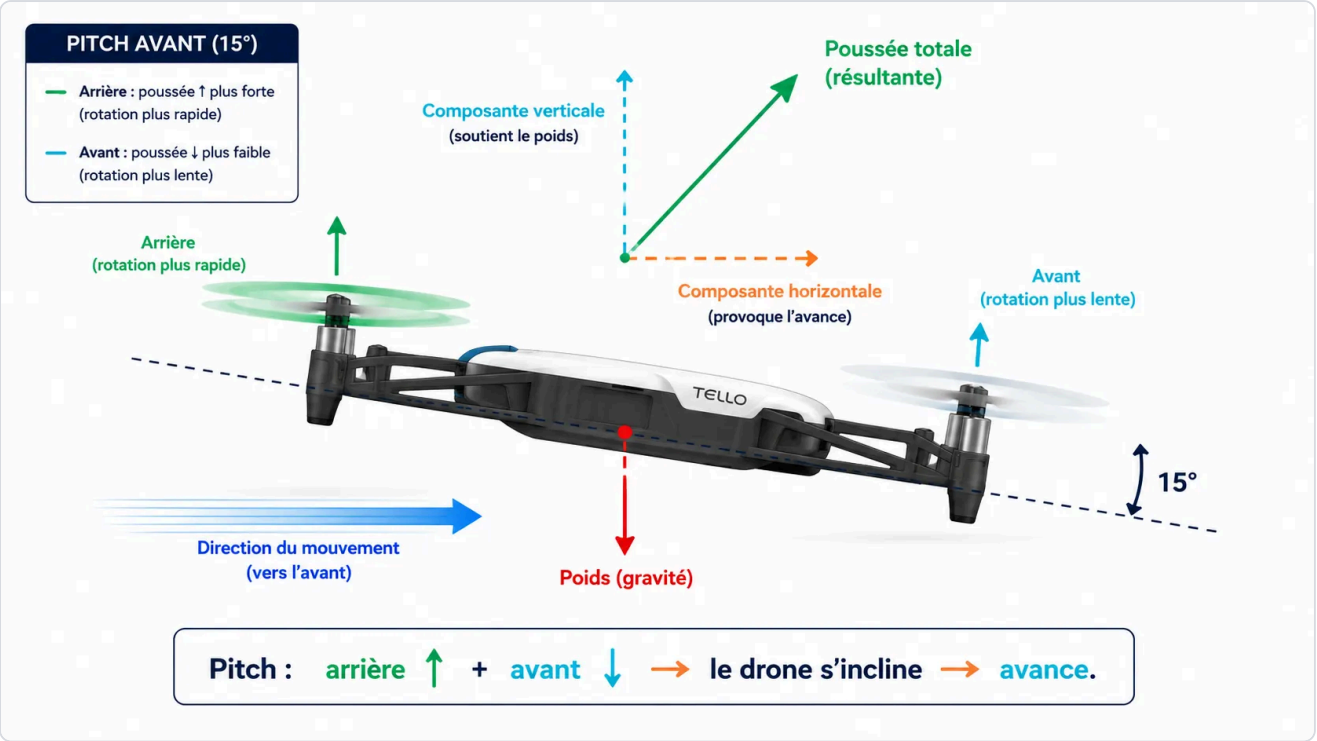
Pour faire tourner le drone à droite : les **2 hélices CW** accélèrent, les **2 hélices CCW** ralentissent. Du coup, le couple "rotation à droite" devient plus fort que le couple "rotation à gauche", et le drone tourne à droite. C'est subtil parce que le drone ne se déplace pas — il pivote, simplement.

Exemples d'utilisation :

- Tu veux que la caméra frontale regarde dans une autre direction → tu pivotes en yaw
- Tu veux faire un demi-tour pour repartir d'où tu viens → yaw de 180°
- Tu fais un selfie aérien → yaw lent pour balayer le paysage

Pitch : avancer et reculer

Le pitch, c'est le mouvement d'inclinaison **vers l'avant ou l'arrière**. C'est cette inclinaison qui décompose la poussée en deux composantes (verticale + horizontale) et permet au drone d'avancer.



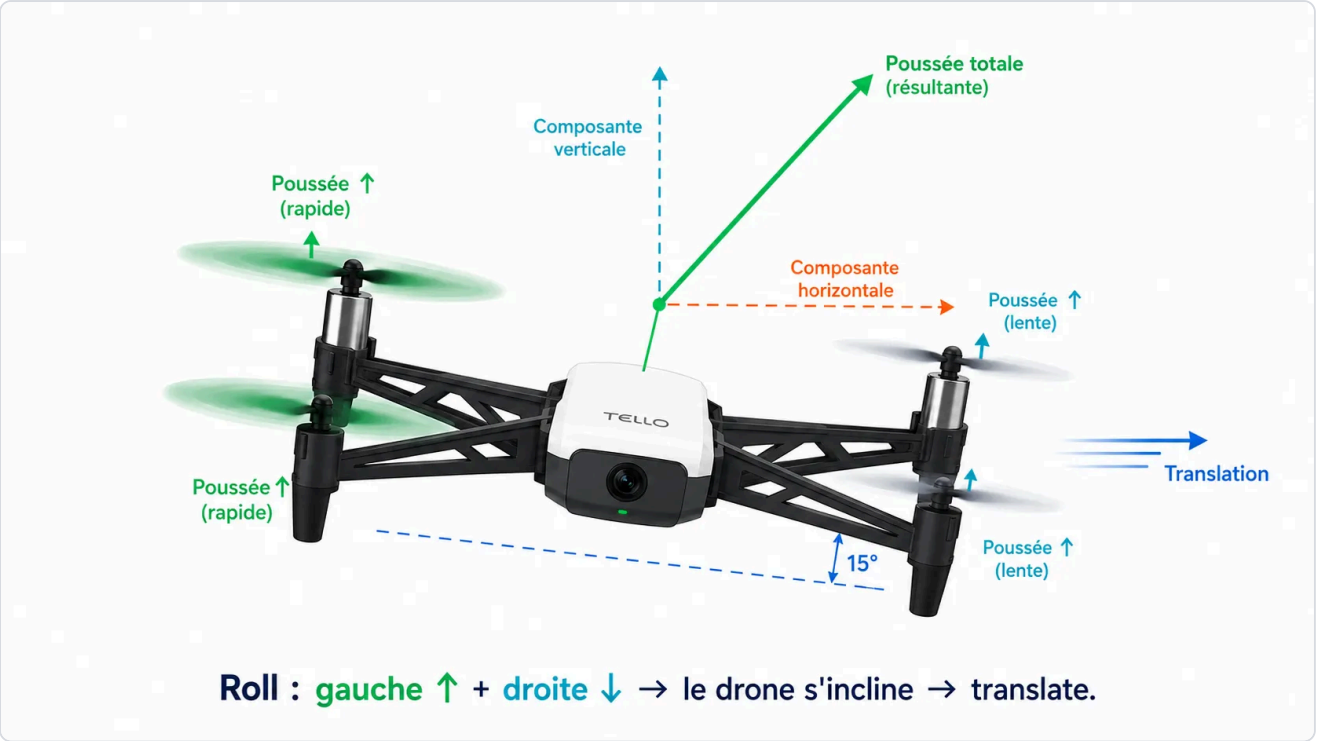
Pour avancer : les **2 hélices arrière accélèrent**, les **2 hélices avant ralentissent**. Le drone s'incline vers l'avant d'un angle θ. La poussée totale se décompose en une force verticale (T × cos θ, qui soutient le drone) et une force horizontale (T × sin θ, qui le fait avancer).

Exemples d'utilisation :

- Tu veux que le drone aille filmer un sujet plus loin → pitch avant
- Tu veux qu'il revienne vers toi → pitch arrière
- Tu veux le faire avancer plus vite → tu inclines plus fort (mais attention à la limite de 30°)

Roll : translater gauche et droite

Le roll, c'est exactement le même principe que le pitch, mais sur l'**autre axe** : inclinaison latérale, vers la gauche ou la droite.



Pour aller à droite : les **2 hélices gauche accélèrent**, les **2 hélices droite ralentissent**. Le drone s'incline à droite, la poussée se décompose, et le drone **translate** vers la droite.

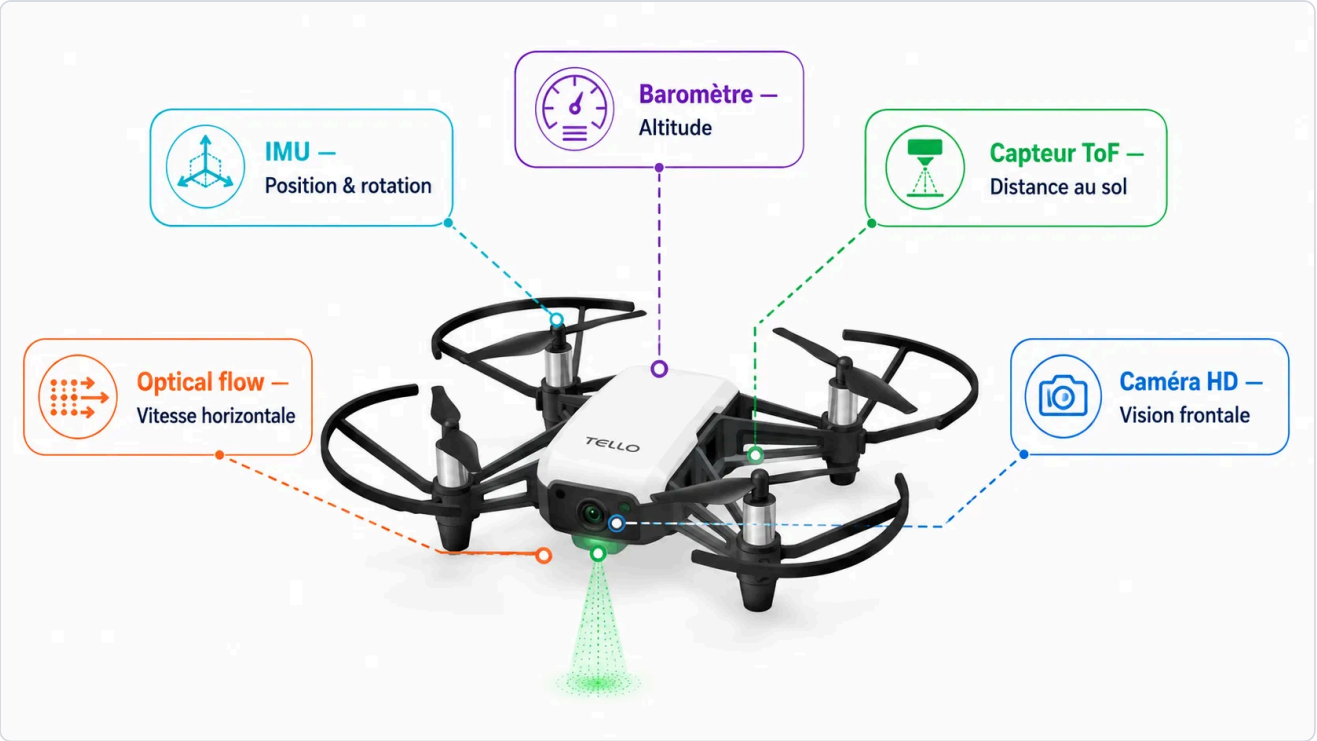
⚠️ **Subtilité importante** : un drone ne peut **pas** se déplacer latéralement sans s'incliner. C'est physiquement impossible. Quand tu vois un drone "glisser" sur le côté, en réalité il est très légèrement incliné — la caméra stabilisée fait juste croire qu'il est droit.

Exemples d'utilisation :

- Tu veux faire un travelling cinéma le long d'un mur → roll
- Tu veux esquiver un obstacle → roll rapide
- Combiné avec un yaw, tu peux tourner autour d'un sujet en gardant la caméra dessus (orbite)

Les capteurs embarqués

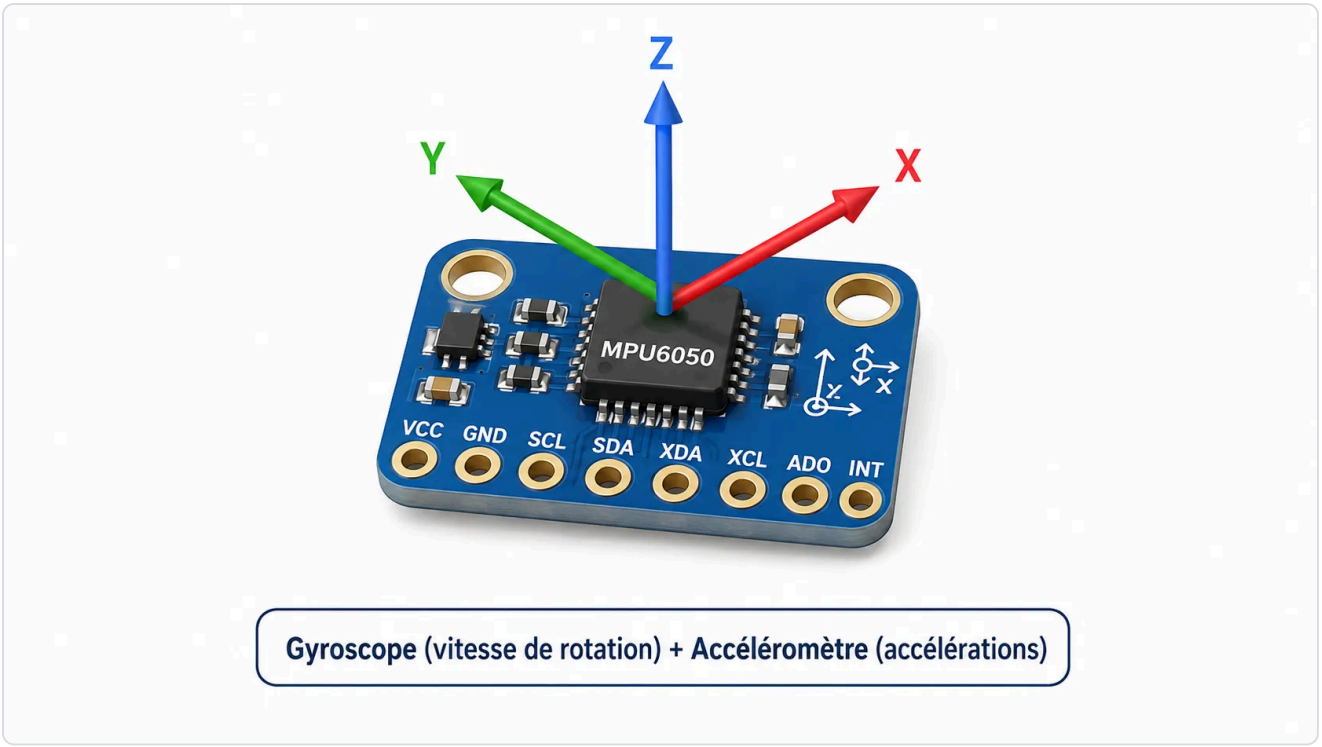
Si le drone est stable, sait où il est, et corrige automatiquement ses dérives, c'est grâce à ses **capteurs**. Le Tello en embarque plusieurs, et chacun a un rôle bien précis.



Capteur 1 : l'IMU (Inertial Measurement Unit)

C'est **le capteur le plus important** du drone. Sans lui, rien ne fonctionne. L'IMU est en réalité **deux capteurs combinés** dans un même petit boîtier (un chip de quelques millimètres) :

- **Gyroscope** : mesure la vitesse de rotation autour de chaque axe (X, Y, Z) en degrés par seconde. Si le drone tourne sur lui-même à 90°/s, le gyroscope le détecte instantanément.
- **Accéléromètre** : mesure les accélérations sur chaque axe (X, Y, Z) en m/s². Si tu pousses le drone, l'accéléromètre détecte l'à-coup. Au repos, l'accéléromètre mesure aussi la gravité (9,81 m/s² vers le bas), ce qui permet au drone de savoir où est "le bas".

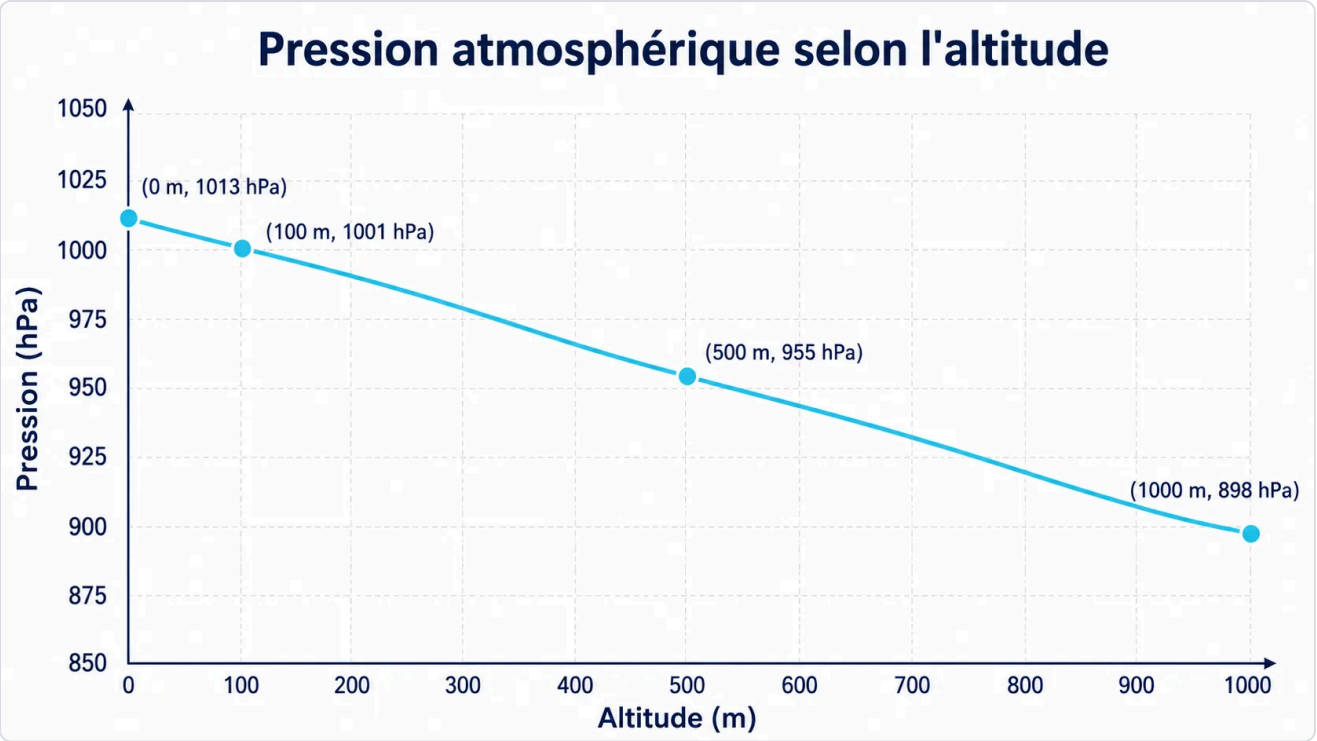


🧠 **Analogie** : c'est l'**oreille interne** humaine. Quand tu fermes les yeux, tu sais quand même si tu penches en avant, en arrière, ou si tu tournes — c'est ton oreille interne qui te le dit. L'IMU joue ce rôle pour le drone, à 1000 Hz (mille mesures par seconde).

Capteur 2 : le baromètre

Le baromètre mesure la **pression atmosphérique**. Comme la pression diminue avec l'altitude (environ 1 hPa de moins tous les 8 mètres), le baromètre permet d'estimer l'altitude.

```yaml
Au sol (Bruxelles)    : 1013 hPa
À 100 m d'altitude    : ~1001 hPa
À 1000 m d'altitude   : ~898 hPa
```



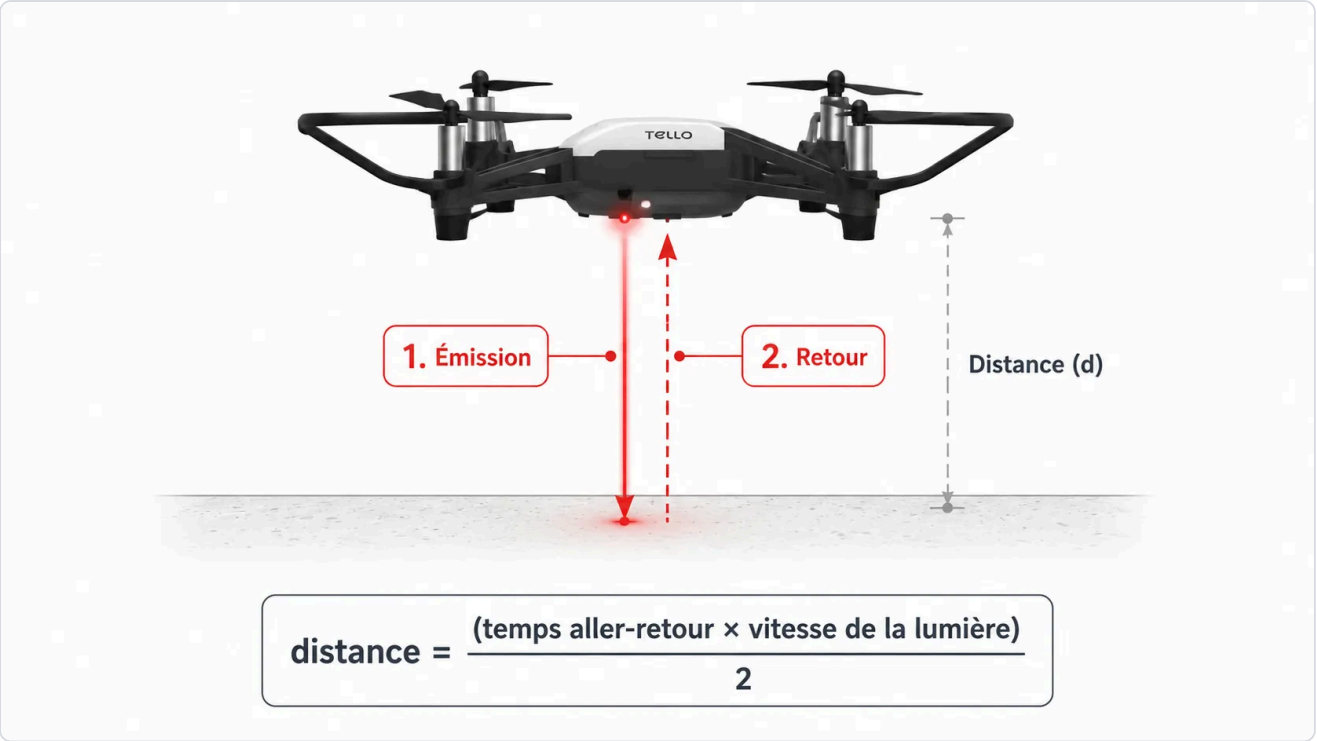
⚠️ **Limitation** : le baromètre est sensible aux changements météo. Si tu vols pendant une journée de chute de pression atmosphérique, ton drone peut croire qu'il monte alors qu'il ne bouge pas. C'est pour ça que le baromètre est **complété** par d'autres capteurs (ToF, optical flow).

Capteur 3 : le capteur ToF (Time of Flight)

Le ToF est un petit laser placé **sous le drone** qui mesure la distance au sol. Le principe est super simple :

1. Le laser émet un **flash de lumière** (invisible, infrarouge)
2. Le flash se réfléchit sur le sol et revient au capteur
3. Le capteur mesure le **temps** que la lumière a mis pour faire l'aller-retour
4. Comme la vitesse de la lumière est connue (300 000 km/s), il calcule la distance

```
distance = (temps aller-retour × vitesse de la lumière) / 2
```

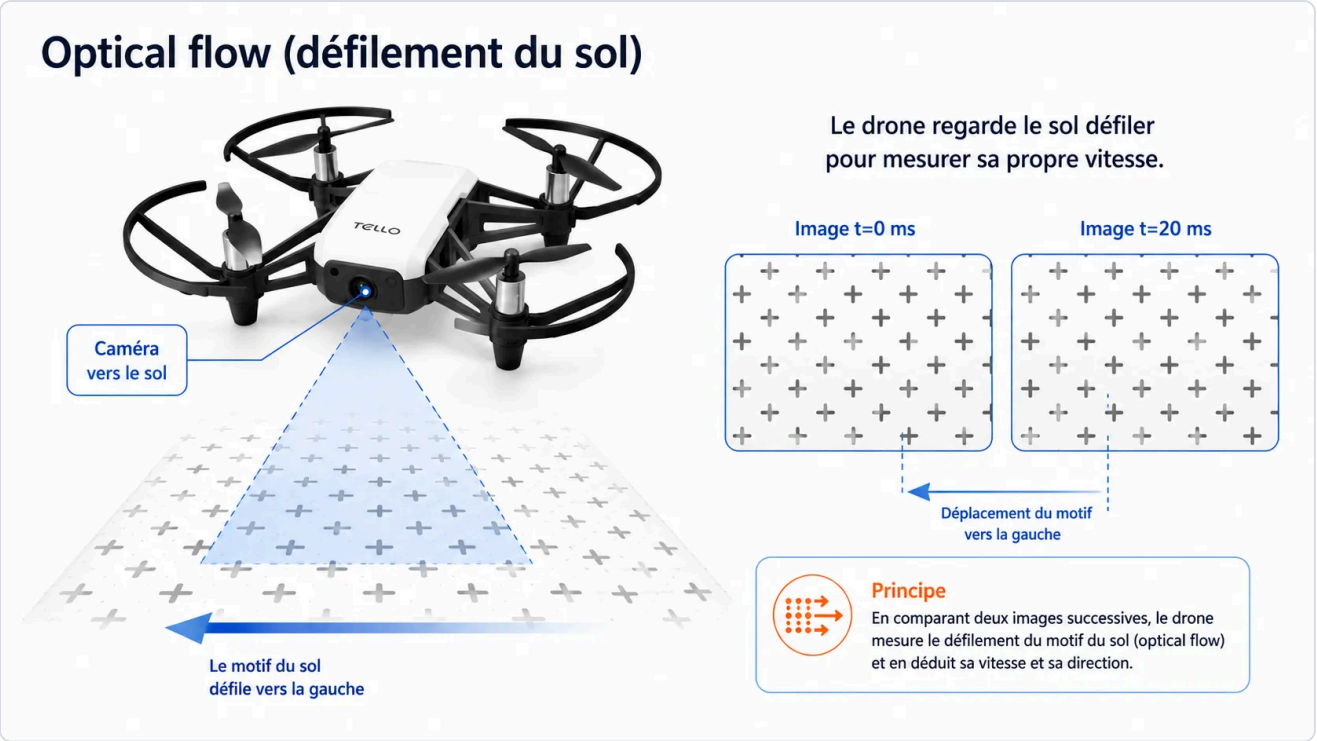


⚡ **Pont électronique direct** : tu te souviens du **HC-SR04** que tu as monté sur le robot Arduino pour détecter les obstacles ? Le ToF du Tello, **c'est exactement le même principe**, sauf qu'il utilise de la **lumière** au lieu du **son**. Avantages : plus rapide (la lumière va beaucoup plus vite que le son), plus précis (le faisceau lumineux est plus fin que le cône d'ultrasons), et plus discret.

Capteur 4 : l'optical flow

L'optical flow est une **petite caméra placée sous le drone** qui filme le sol en permanence. Elle ne sert pas à enregistrer des vidéos — elle sert à **détecter le mouvement** du drone.

Le principe : la caméra capture une image, puis une autre quelques millisecondes plus tard, et **compare les deux**. Si les motifs du sol ont défilé vers la gauche entre les deux images, c'est que le drone s'est déplacé vers la droite. Si les motifs ont défilé vite, c'est que le drone bouge vite.



🤖 **Pont EV3 direct** : tu te souviens, sur le robot Mindstorm, on utilisait les **encodeurs** des roues pour savoir combien le robot avait avancé (1 rotation = 360 degrés moteur = X centimètres au sol). Le drone n'a **pas de roue**, donc il ne peut pas compter de tours. Mais il a **besoin de la même information** (savoir de combien il s'est déplacé). Solution : il **regarde** le sol bouger. Même problème, capteur différent.

⚠️ **Limitation** : l'optical flow a besoin d'un sol avec des **motifs**. Au-dessus d'un sol blanc uniforme, ou de l'eau, il "voit" rien et perd la trace. C'est pour ça qu'on conseille de voler avec un Tello sur du parquet, du carrelage, ou de la moquette à motifs — pas sur du linoléum blanc.

Capteur 5 : la caméra HD frontale

La grande caméra à l'avant du Tello sert à filmer ou à streamer la vidéo en temps réel. Sur le Tello standard, elle film en 720p à 30 images par seconde, et envoie le flux en WiFi à ton smartphone ou ton script Python.



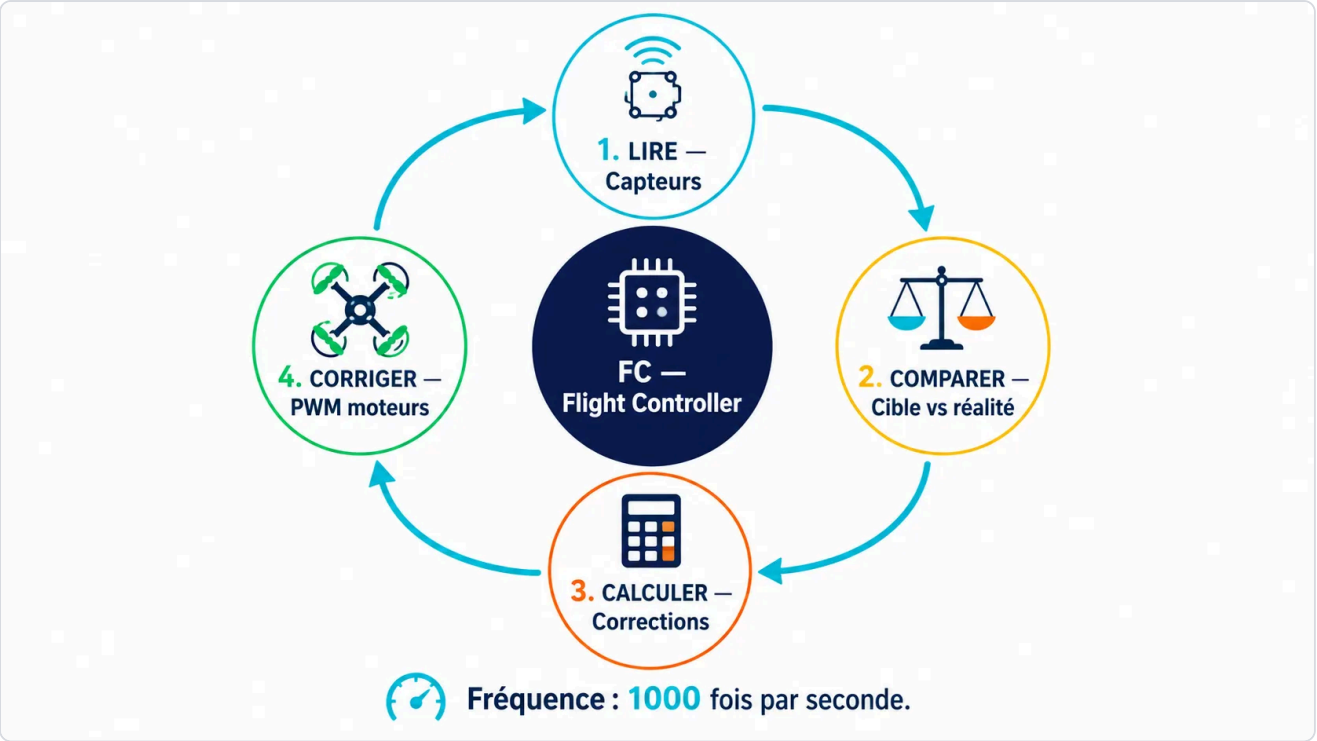
Pour les séances suivantes, on s'en servira pour faire de la **reconnaissance d'objets ou de couleurs** avec OpenCV en Python. C'est ce qui permet à des drones plus avancés de **suivre automatiquement** un sujet (mode "Follow Me"), de détecter des panneaux, ou de reconnaître des visages.

La boucle de contrôle

On arrive au concept le plus important du chapitre. Sans ce mécanisme, **rien de ce qu'on vient de voir ne fonctionnerait**.

Le Flight Controller du drone exécute en permanence la boucle suivante :

```
boucle infinie :
    1. LIRE les capteurs        (IMU, baromètre, ToF, optical flow)
    2. COMPARER avec l'objectif (où je veux être / comment je veux être orienté)
    3. CALCULER les corrections (de combien je dois ajuster chaque moteur)
    4. CORRIGER en envoyant les nouvelles vitesses PWM aux 4 ESC
fin boucle
```



Tu reconnais ?

⚡ **Pont Arduino magistral** : c'est **exactement** la structure de ton `loop()` Arduino. Souviens-toi du robot évitement d'obstacles :

```
void loop() {
    int distance = lireUltrason();     // 1. LIRE le capteur
    if (distance < 20) {               // 2. COMPARER avec l'objectif
        tournerDroite();               // 3. CALCULER la correction
    } else {                           //
        avancer();                     // 4. AGIR (envoyer aux moteurs)
    }
}
```

C'est exactement la même architecture. Sauf que :

Robot Arduino 2WD	Drone Tello
`loop()` exécuté ~100 fois/seconde	Boucle exécutée **1000 fois/seconde**
Lit 1 capteur (HC-SR04)	Lit ~10 capteurs simultanément
Contrôle 2 moteurs DC	Contrôle 4 moteurs brushless
Si plante, tombe à l'arrêt	Si plante, **tombe** tout court

Pourquoi 1000 Hz ?

Cette fréquence élevée — **1000 corrections par seconde** — c'est ce qui rend un drone **stable**. Quand tu lâches les commandes, le drone ne dérive pas pendant 5 secondes avant de réagir : il se corrige en permanence, à une vitesse imperceptible pour l'œil humain.

C'est aussi pourquoi un drone moderne est **facile à piloter** : tu donnes une consigne ("avance à 1 m/s"), et le drone gère **tout seul** les milliers de petits ajustements nécessaires pour la respecter (compensation du vent, dérive, inclinaison). Toi, tu pilotes une **intention**. Lui, il fait l'**arithmétique**.

Synthèse

Ce qu'il faut retenir avant de toucher un Tello :

1. **Un drone vole parce qu'il génère en permanence une force vers le haut** égale ou supérieure à son poids (≈ 0,78 N pour le Tello).
2. **4 hélices, 2 sens de rotation** (2 CW + 2 CCW) : pour annuler les couples de réaction et garder la stabilité.
3. **4 commandes de pilotage** (throttle, yaw, pitch, roll) obtenues en modulant différemment les 4 vitesses moteurs.
4. **Les déplacements horizontaux passent toujours par une inclinaison** (pitch ou roll), qui décompose la poussée en deux composantes via sin/cos.
5. **5+ capteurs embarqués** : IMU (oreille interne du drone), baromètre (altitude), ToF (distance au sol), optical flow (vitesse horizontale), caméra HD (vidéo).
6. **Une boucle de contrôle à 1000 Hz** qui lit, compare, calcule et corrige — c'est le `loop()` Arduino sous stéroïdes.

Et maintenant : à la pratique

Tu as compris la théorie. La meilleure façon de l'ancrer, c'est de **piloter**. Dans la deuxième partie de cette séance, tu vas, en binôme :

Étape 1 — Installer l'application Tello

- Sur ton smartphone (Android ou iOS), va sur le Play Store ou l'App Store
- Recherche **"Tello"** (ou **"Tello EDU"** pour la version plus avancée)
- Installe l'app officielle (icône blanche avec un drone bleu)

Étape 2 — Connecter ton drone

- Allume le drone Tello (interrupteur sur le côté, près de la batterie)
- La LED du drone clignote en jaune → le drone est en mode "WiFi disponible"

- Sur ton smartphone, va dans les **paramètres WiFi**
- Cherche un réseau qui s'appelle `TELLO-XXXXXX` (les X = ID unique du drone)
- Connecte-toi à ce réseau (pas de mot de passe)
- Ouvre l'app Tello : elle détecte automatiquement le drone connecté

Étape 3 — Checklist sécurité avant décollage

Avant de faire décoller le drone, **toujours** vérifier :

- ✅ Zone dégagée d'au moins 2 mètres autour du drone
- ✅ Personne ne tient ses doigts près des hélices
- ✅ Batterie suffisante (au moins 30%)
- ✅ Sol dégagé sous le drone (le ToF en a besoin)
- ✅ Pas de vent (on est en intérieur, donc ok)
- ✅ Tu sais où est le **bouton d'atterrissage d'urgence** dans l'app

Étape 4 — Premier décollage

- Appuie sur le bouton **"Décoller"** dans l'app
- Le drone monte automatiquement à 1 mètre et stationne
- Teste **doucement** les 4 commandes une par une avec les sticks
- Quand tu as fini : appuie sur **"Atterrir"**

Étape 5 — Le défi du jour : course par équipes 🏆

Une fois que ton binôme maîtrise les 4 commandes, place au **défi**.

Règles :

1. Chaque binôme **conçoit un parcours** dans la zone de vol :
   - Un point de départ (cerceau au sol, par exemple)
   - Un point d'arrivée (autre cerceau)
   - **Au moins 3 obstacles** entre les deux (chaises, cartons, lignes au sol à franchir, anneaux à traverser)
   - Le parcours doit utiliser **les 4 commandes** au minimum (pas juste avancer)
2. Une fois le parcours validé par le prof, **les autres binômes viennent le tester** :
   - Un membre du binôme conçoit son parcours et l'explique aux autres
   - Chaque autre binôme essaie son parcours, **un essai chacun**
   - Le **chronomètre** démarre quand le drone décolle, s'arrête quand il atterrit dans la zone d'arrivée

- **Pénalité** : crash ou contact avec un obstacle = +30 secondes ajoutées au chrono
- **Disqualification** : abandon ou drone qui sort de la zone de vol

3. **Classement général** :
   - Le binôme qui a fait **le meilleur chrono moyen** sur l'ensemble des parcours qu'il a testés gagne le défi
   - Bonus : titre de **"meilleur designer de parcours"** au binôme dont le parcours a été jugé le plus créatif/équilibré par les autres

🏆 Le binôme gagnant repart avec... **la fierté de l'avoir fait** (et probablement un statut légendaire dans le groupe Discord du cours).

> 💡 ***Conseil pour les designers de parcours*** *: un bon parcours, ce n'est pas un parcours* ***impossible****. C'est un parcours* ***techniquement varié*** *qui force à utiliser les 4 commandes intelligemment. Un parcours qui demande juste d'aller tout droit, c'est ennuyeux. Un parcours qui demande de passer dans un anneau, faire un yaw, descendre, repasser, c'est fun.*

Prochaine séance

La séance prochaine, on remplace le smartphone par du **Python**. Tu réaliseras que piloter et programmer, c'est presque la même chose : dans les deux cas, tu envoies des commandes au Flight Controller. La seule différence, c'est **qui les compose** : toi avec tes pouces sur les sticks, ou ton script Python qui décide à ta place.

Prêt à décoller ? 🚀Drones

...

5.3 Premiers Pas avec Python

> 🎯 ***Objectif du chapitre*** *: À la fin de ce chapitre, tu auras installé Python sur ta machine, tu sauras écrire et exécuter un programme simple, et tu maîtriseras les briques de base (variables, affichage, attente, modules) qui te serviront pour piloter ton drone au chapitre suivant.*

🗺️ Ce qu'on va apprendre aujourd'hui

Leçon	Titre	Tu sauras faire...
1	Installer Python et Thonny	Mettre Python en place sur ton PC
2	Mon premier programme	Afficher du texte, exécuter du code
3	Les variables	Stocker et réutiliser des valeurs

Leçon	Titre	Tu sauras faire...
4	Parler à l'utilisateur	Demander une info et y répondre
5	Attendre et importer	Mettre des pauses, utiliser des modules

Temps estimé : 45 min à 1h de pratique

🧠 Avant de Commencer : Python, c'est quoi ?

Tu as déjà programmé avec **Scratch** (blocs colorés à empiler), avec **Processing** (du code pour dessiner), avec **les briques EV3** (programmation visuelle pour le robot). Toutes ces interfaces, c'était de la programmation. Python aussi.

La différence ? **Python s'écrit en texte**. Pas de blocs à glisser : tu tapes des instructions ligne par ligne, et l'ordinateur les exécute dans l'ordre.

Pourquoi Python ?

🌍 Dans le monde réel	🤖 Dans ce cours
C'est le langage le plus utilisé en data science et IA	On va l'utiliser pour piloter le drone Tello
Il sert chez Google, Netflix, la NASA, l'ESA...	Sa syntaxe est lisible — proche du français en pseudocode
Il est gratuit, multi-plateforme, et a une énorme communauté	Tu peux installer plein de modules pour faire plein de choses

Bridge avec Scratch

Tout ce que tu as vu en Scratch existe aussi en Python. C'est juste écrit différemment.

En Scratch	En Python
🟦 Bloc `dire "Bonjour"`	`print("Bonjour")`
🟧 Bloc `attendre 1 seconde`	`time.sleep(1)`
🟪 Bloc `variable = 10`	`variable = 10`
🟨 Bloc `si... alors...`	`if ... :`
🟩 Bloc `répéter 4 fois`	`for i in range(4):`

Tu connais déjà les concepts. Reste à apprendre la **manière d'écrire** en Python.

📍 Leçon 1 : Installer Python et Thonny

Pour écrire du code Python, on a besoin de deux choses :

1. **Python lui-même** — c'est le moteur qui va lire et exécuter ton code
2. **Un éditeur de code** — c'est le logiciel dans lequel tu vas taper ton programme

On va utiliser **Thonny**, un éditeur fait spécialement pour les débutants Python. Il est léger, gratuit, et il intègre déjà tout ce qu'il faut.

Étape 1 : Installer Python

1. Va sur python.org/downloads
2. Clique sur le gros bouton jaune **"Download Python 3.x"**
3. Lance l'installateur
4. Clique sur **Install Now** et attends que ça finisse

💡 **Astuce** : sur Windows, l'installateur récent fait tout seul ce qu'il faut (notamment l'ajout au PATH). Si jamais une commande Python ne fonctionne pas plus tard depuis un terminal, désinstalle puis réinstalle en cochant l'option **"Add Python to PATH"** sur l'écran principal de l'installateur.

Étape 2 : Installer Thonny

1. Va sur thonny.org
2. Télécharge la version pour ton système (Windows, Mac ou Linux)
3. Installe normalement, "Suivant suivant suivant"

Étape 3 : Vérifier que ça marche

1. Lance **Thonny**
2. Tu vois deux zones :
   - En haut, **l'éditeur** (zone blanche où tu vas taper ton code)
   - En bas, la **console** (zone où s'affichent les résultats)
3. Dans la console, après les `>>>`, tape : `1 + 1` puis appuie sur **Entrée**
4. Si tu vois `2` s'afficher, **c'est gagné !** 🎉

⚠️ Que faire si...

Le problème	La solution
`python` n'est pas reconnu dans le terminal	Tu as oublié de cocher "Add Python to PATH". Désinstalle et réinstalle.
Thonny refuse de s'ouvrir	Vérifie que tu as bien installé Python AVANT Thonny

Le problème	La solution
La console affiche un message bizarre	Note-le et lève la main, on regarde ensemble

📍 Leçon 2 : Mon Premier Programme

Bon. Python est installé. Thonny tourne. Maintenant on code.

Hello, world!

Toute la communauté des programmeurs partage une tradition : le **premier programme qu'on écrit dans un nouveau langage affiche le mot "Hello, world!"**. C'est un rite de passage.

Dans l'éditeur Thonny (la zone du haut), tape **exactement** ceci :

```python
print("Hello, world!")
```

Puis appuie sur le bouton ▶ **Exécuter** (ou la touche **F5**).

Thonny va te demander d'enregistrer le fichier. Donne-lui un nom (ex: `hello.py`) et un dossier. **L'extension** `.py` veut dire "fichier Python".

Tu devrais voir ceci s'afficher dans la console :

```
Hello, world!
```

🎉 **Tu viens d'exécuter ton premier programme Python.**

Qu'est-ce qu'on vient de faire ?

Décortiquons cette ligne minuscule mais riche en concepts :

```python
print("Hello, world!")
```

Élément	Ce que c'est
`print`	Le nom d'une **fonction** intégrée à Python. Elle affiche du texte.
`(` et `)`	Les parenthèses indiquent ce qu'on **donne** à la fonction (ses "ingrédients").
`"Hello, world!"`	Un **texte** (on dit aussi une "chaîne de caractères" ou "string"). Les guillemets indiquent que c'est du texte.

💡 **Analogie** : `print` est comme une machine à café. Tu lui donnes un ingrédient (le texte), tu appuies sur le bouton (▶), elle te sort un résultat (le texte affiché).

✏️ Mini-exercice 🥉 Bronze

Modifie ton programme pour qu'il affiche :

```bash
Bonjour !
Je m'appelle Python.
J'ai très envie de piloter un drone.
```

💡 **Indice** : Une ligne `print(...)` n'affiche qu'une seule ligne de texte. Combien de `print` te faut-il ?

📍 Leçon 3 : Les Variables

Imagine que tu doives afficher ton prénom 10 fois dans ton programme. Tu pourrais écrire :

```python
print("Bonjour Thibault")
print("Salut Thibault")
print("Coucou Thibault")
# ... encore 7 fois
```

Mais si tu changes de prénom ? Tu dois tout réécrire. Et si tu te trompes une fois, ton programme devient incohérent.

Solution : on stocke ton prénom dans une **variable**.

Une variable, c'est quoi ?

C'est une **boîte avec une étiquette**. Tu mets quelque chose dedans, tu donnes un nom à la boîte, et après tu peux y faire référence par ce nom.

```python
prenom = "Thibault"

print("Bonjour")
print(prenom)
print("Comment vas-tu ?")
```

Exécute. Tu obtiens :

```
Bonjour
Thibault
Comment vas-tu ?
```

Maintenant change la première ligne en `prenom = "Marie"` et relance. Le programme entier s'adapte sans rien d'autre à modifier. **C'est ça, le pouvoir des variables.**

Les règles de nommage

Quelques règles pour nommer une variable en Python :

✅ Autorisé	❌ Interdit
`prenom`	`prénom` (pas d'accents)
`prenom_user`	`prenom user` (pas d'espace)
`prenom2`	`2prenom` (pas commencer par un chiffre)
`MaVariable`	`if` , `print` , `for` … (mots réservés Python)

💡 **Convention** : en Python, on écrit les noms de variables en **minuscules avec des underscores** : `ma_variable` , `prix_total` , `nombre_de_drones` . Lisible et conforme aux usages.

Les types de valeurs

Une variable peut contenir différentes sortes de choses :

PYTHON

```python
prenom = "Thibault"          # un texte (string)
age = 35                     # un nombre entier (integer)
hauteur_drone = 1.50         # un nombre à virgule (float)
est_majeur = True            # un booléen (True ou False)
```

⚠️ **Attention aux guillemets** : les textes sont entre `"..."` , les nombres NON.

PYTHON

```python
nombre = 10        # un vrai nombre, on peut faire des maths avec
nombre = "10"      # un TEXTE qui ressemble à un nombre — on ne peut PAS faire de
maths avec
```

Faire des calculs

Tu peux utiliser les variables comme dans une calculatrice :

```python
prix_drone = 120
nombre = 8
total = prix_drone * nombre
print(total)
```

Résultat :

```
960
```

Opérateur	Effet	Exemple
`+`	Addition	`5 + 3` → `8`
`-`	Soustraction	`5 - 3` → `2`
`*`	Multiplication	`5 * 3` → `15`
`/`	Division	`10 / 3` → `3.333...`
`//`	Division entière	`10 // 3` → `3`
`%`	Reste de la division	`10 % 3` → `1`

✏️ Mini-exercice 🥈 Argent

Écris un programme qui calcule le **prix total de ta commande de matériel pour la classe** :

- 8 drones Tello à 120€
- 8 batteries supplémentaires à 25€
- 4 hubs de chargement à 35€

Le programme doit afficher le total. Utilise des variables avec des noms parlants.

📍 Leçon 4 : Parler à l'Utilisateur

Pour l'instant, tes programmes affichent toujours la même chose. C'est statique. Pour les rendre **interactifs**, on va apprendre à **demander** des informations à l'utilisateur.

La fonction `input()`

```python
prenom = input("Comment tu t'appelles ? ")
print("Salut", prenom, "!")
```

Quand tu exécutes :

```
Comment tu t'appelles ?
```

Le programme **attend** que tu tapes une réponse. Tape ton prénom, appuie sur Entrée :

```
Comment tu t'appelles ? Thibault
Salut Thibault !
```

Comment ça marche ?

- `input("...")` affiche le texte entre guillemets, puis attend la réponse de l'utilisateur
- Ce que l'utilisateur tape est **stocké dans la variable** à gauche du `=`
- Ce qu'on stocke est **toujours du texte** (un string)

⚠️ Le piège des nombres

Si tu fais ça :

PYTHON

```python
nombre = input("Donne-moi un nombre : ")
double = nombre * 2
print(double)
```

Et que tu tapes `5`, tu obtiens... `55`. Quoi ?!

Explication : pour Python, ce que tu as tapé est le **texte** `"5"`, pas le nombre `5`. Multiplier un texte par 2, ça le répète deux fois (`"5" * 2` = `"55"`).

Pour avoir un vrai nombre, il faut **convertir** :

PYTHON

```python
nombre = int(input("Donne-moi un nombre : "))
double = nombre * 2
print(double)
```

`int(...)` veut dire "convertis ceci en **int**eger (nombre entier)". Maintenant si tu tapes `5`, tu obtiens `10`. ✅

Fonction	Convertit en...
`int(x)`	Nombre entier
`float(x)`	Nombre à virgule

Fonction	Convertit en…
`str(x)`	Texte

✏️ Mini-exercice 🥈 Argent

Écris un programme qui :

1. Demande son prénom à l'utilisateur
2. Demande son année de naissance
3. Calcule son âge approximatif (année actuelle - année de naissance)
4. Affiche : `"Bonjour [prénom], tu as environ [âge] ans."`

💡 **Indice 1** : tu auras besoin de `int(...)` quelque part.

💡 **Indice 2 — la question délicate** : comment ton programme va-t-il connaître l'année actuelle ?

Deux approches s'offrent à toi :

- **Solution facile mais moche** : écris dans ton code `annee_actuelle = 2026`. Ça marche… jusqu'au 1er janvier prochain où ton programme sera faux ! Il faudra le modifier chaque année.
- **Solution élégante** : Python a un module qui s'occupe des dates. Tu en connais déjà un dans la même famille — celui qu'on a importé pour `sleep` au-dessus. Cherche `datetime` (= "date et heure") et essaie de trouver comment récupérer l'année courante. Si tu sèches, Google "python current year" et la première réponse te dépannera. La solution tient en 2 lignes.

Indice gratuit pour t'orienter : il y a un objet `datetime.date.today()` qui te donne la date d'aujourd'hui, et tu peux accéder à son année avec un attribut nommé… bah devine.

📍 Leçon 5 : Attendre, et Utiliser des Modules

Deux concepts qu'on va beaucoup utiliser pour le drone : **mettre des pauses dans le code** et **importer des outils** que d'autres ont écrits.

Mettre une pause

En Scratch, tu avais le bloc 🟧 `attendre 1 seconde`. En Python, c'est la fonction `sleep` du module `time`.

```python
import time

print("3...")
time.sleep(1)
print("2...")
time.sleep(1)
print("1...")
time.sleep(1)
print("Décollage ! 🚀")
```

Exécute. Tu vois bien la pause d'une seconde entre chaque ligne.

C'est quoi, un module ?

Python tout seul, c'est une boîte à outils de base. Mais beaucoup de gens ont écrit des **outils supplémentaires** rangés dans des **modules**. Quand tu écris :

```python
import time
```

Tu dis à Python : *"emprunte-moi la boîte à outils* `time`, j'en ai besoin".

💡 **Analogie** : Python sans modules = une trousse de base (stylos, crayons, gomme). Importer un module = ouvrir un tiroir spécialisé (la trousse de géométrie, la trousse de chimie...).

Une fois importé, tu accèdes aux outils du module avec `nom_du_module.nom_de_l_outil` :

Code	Ce que ça fait
`time.sleep(2)`	Attendre 2 secondes
`time.sleep(0.5)`	Attendre 0,5 seconde (une demi-seconde)
`time.sleep(0.1)`	Attendre 0,1 seconde

Préviewing : un module qu'on va beaucoup utiliser

Au chapitre suivant, on importera le module `djitellopy` pour piloter le drone :

```python
from djitellopy import Tello

drone = Tello()
drone.connect()
drone.takeoff()
```

Tu ne comprends pas encore tout, c'est normal. On verra ça en détail. Mais déjà tu reconnais : un `import`, puis on utilise les outils du module. C'est exactement le même principe que `time.sleep`.

✏️ Mini-exercice 🥇 Or

Écris un programme **compte à rebours décollage drone** qui :

1. Demande à l'utilisateur le prénom du pilote
2. Affiche `"Bienvenue [prénom], préparation au décollage."`
3. Attend 2 secondes
4. Affiche `"5..."`, attend 1 seconde, puis `"4..."`, etc. jusqu'à `"1..."`
5. Affiche `"🚀 Décollage !"`

💡 **Astuce** : tu peux soit écrire 5 fois `print` + `sleep`, soit (si tu te sens téméraire) chercher comment faire une boucle. On verra les boucles au chapitre suivant — pas obligé de plonger maintenant.

✏️ Mini-exercice 💎 Diamant

Reprends le programme précédent et améliore-le :

1. Demande à l'utilisateur le **nombre de secondes** du compte à rebours
2. Le programme s'adapte (`compte_a_rebours = int(input(...))`)
3. Bonus : si l'utilisateur tape `0` ou un nombre négatif, affiche directement `"Décollage immédiat !"`

💡 **Indice** : on n'a pas encore vu les conditions `if/else`. Si tu veux tenter, cherche par toi-même ou demande au prof — on verra ça au chapitre suivant en détail.

🎯 Auto-évaluation

Niveau	J'ai réussi à...
🥉 Bronze	Installer Python + Thonny, et afficher du texte avec `print`
🥈 Argent	Utiliser des variables, faire des calculs, lire l'input utilisateur
🥇 Or	Faire un compte à rebours avec `time.sleep` et un module importé
💎 Diamant	Adapter le compte à rebours en fonction de l'input utilisateur

📚 Vocabulaire à Retenir

Terme	Définition
Python	Langage de programmation texte, célèbre pour sa simplicité et sa polyvalence
Thonny	Éditeur de code spécialement conçu pour les débutants Python
`.py`	Extension des fichiers Python
Console	Zone où le programme affiche ses résultats
`print()`	Fonction qui affiche du texte
`input()`	Fonction qui demande une saisie à l'utilisateur
Variable	Boîte étiquetée qui stocke une valeur réutilisable
String / chaîne	Du texte, écrit entre guillemets `"..."`
Integer / int	Nombre entier sans virgule
Float	Nombre à virgule
Module	Boîte à outils Python à importer pour bénéficier de ses fonctions
`import`	Mot-clé qui charge un module
`time.sleep(n)`	Met le programme en pause pendant `n` secondes
`datetime`	Module Python qui gère les dates et les heures

🚀 Pour Aller Plus Loin

Si tu as fini avant les autres :

- **Calculatrice perso** : programme qui demande deux nombres et une opération (+, -, *, /), et affiche le résultat
- **Convertisseur** : programme qui convertit des euros en dollars (ou inversement) selon un taux que l'utilisateur saisit
- **Mad Libs** : programme qui demande des mots (un nom, un adjectif, un verbe...) et construit une histoire farfelue en les utilisant dans un texte
- **Compteur d'anniversaire** *(plus avancé, voir indices ci-dessous)* : combien de jours avant ton prochain anniversaire ?
- **Site Python officiel** : docs.python.org/fr/3/tutorial/ — le tutoriel officiel, en français

💡 Indices pour le "Compteur d'anniversaire"

L'idée est simple à formuler :

1. Demande à l'utilisateur **le jour et le mois** de sa date de naissance (jour 1-31, mois 1-12)
2. Récupère **la date d'aujourd'hui** (tu as déjà vu comment dans l'exo précédent)
3. **Calcule combien de jours** séparent aujourd'hui du prochain anniversaire
4. Affiche le résultat

C'est plus subtil que ça en a l'air. Voici les pistes :

- Le module `datetime` peut créer une **vraie date** comme ceci : `datetime.date(2026, 11, 15)` représente le 15 novembre 2026
- Tu peux **soustraire deux dates** entre elles. Le résultat est un objet appelé `timedelta` qui a un attribut `.days` (= nombre de jours d'écart)
- **Attention au cas piégeant** : si la personne est née le 15 mars et qu'on est le 20 mars, son prochain anniversaire est-il en 2026 ou en 2027 ? Ton programme doit gérer ça avec une condition (`if`). Tu n'as pas encore vu les `if` mais tu peux deviner — ou laisser ça pour le prochain chapitre où on les voit en détail.

Tu n'auras pas la solution toute mâchée : à toi d'expérimenter, de chercher dans la doc, et de demander de l'aide si tu coinces. **C'est comme ça qu'on apprend vraiment à programmer.** ✊

5.4 Chapitre : Pilote ton Drone Tello avec Python

> 🎯 ***Objectif du chapitre*** *: À la fin de ce chapitre, tu sauras connecter un drone DJI Tello à ton PC, le piloter par programmation Python, et réaliser une mission complète de décollage / déplacement / atterrissage en autonomie. Tu auras croisé en route les concepts de boucle, de condition et de fonction.*

🗺️ Ce qu'on va apprendre aujourd'hui

Leçon	Titre	Médaille	Tu sauras faire…
0	Installation et préparation	—	Installer `djitellopy` et connecter le drone
1	Premiers vols	🥉 Bronze	Décoller, déplacer, atterrir
2	Structurer ton code	🥈 Argent	Utiliser des variables et des boucles
3	Du code qui décide	🥇 Or	Conditions et fonctions
4	Défi final	💎 Diamant	Mission atterrissage de précision

Temps estimé : 2h30 à 3h de pratique (en binôme)

Prérequis : Avoir suivi le chapitre **"Premiers Pas avec Python"** (installation Python + Thonny, variables, `print`, `input`, `time.sleep`, `import`).

🧠 Avant de Commencer : Comprendre le Tello

Le Tello, c'est quoi ce drone ?

Le **DJI Tello** est un petit drone de 80g, conçu pour apprendre la programmation et la robotique aérienne. Il est petit, robuste, et c'est exactement pour ça qu'on l'utilise en cours.

Caractéristique	Valeur
Poids	80 g
Autonomie	~13 min de vol
Caméra	720p (HD)
Portée	~100 m
Prix	~120 €

Le bridge avec ce qu'on a vu avant

Tu as déjà programmé un robot EV3. Il y a une grosse différence avec le Tello dans la **façon dont il reçoit ton code** :

🤖 Avec l'EV3	🛸 Avec le Tello
Tu écris ton programme sur le PC	Tu écris ton programme sur le PC
Tu **téléverses** le programme dans la brique EV3	Tu **lances** ton programme sur le PC
Le robot **exécute le programme tout seul**	Le PC **envoie les ordres au drone en direct par WiFi**
Le PC peut être éteint après téléversement	Le PC doit rester allumé et connecté tout le temps

C'est deux philosophies différentes :

- L'EV3 a sa propre mémoire et son propre processeur : il devient autonome
- Le Tello est plus un "esclave obéissant" — il fait ce que le PC lui dit, en temps réel

💡 **Analogie** : l'EV3 c'est comme apprendre une chorégraphie par cœur à un danseur. Le Tello c'est comme jouer aux marionnettes — tu manipules en direct.

Comment le drone reçoit nos ordres

Le Tello est aussi un **point d'accès WiFi**. Quand tu l'allumes, il crée son propre réseau WiFi (genre `TELLO-AB1234`). Ton PC s'y connecte, et ensuite, **chaque ligne Python que tu écris envoie un petit message WiFi au drone** qui exécute la commande.

```
 Ton PC                                Le drone

drone.takeoff()  ———[WiFi]———▶   🛸 prend de l'altitude
drone.forward(50) ———[WiFi]———▶   🛸 avance de 50 cm
drone.land()     ———[WiFi]———▶   🛸 atterrit
```

C'est élégant et simple. Allons-y.

📍 Leçon 0 : Installation et Préparation

Étape 1 : Installer `djitellopy` (le module qui parle au drone)

`djitellopy` est la librairie Python qui sait communiquer avec le Tello. C'est elle qu'on va importer dans nos programmes. **Bonne nouvelle** : Thonny te permet d'installer des modules sans ouvrir de terminal.

1. Ouvre **Thonny**
2. Va dans le menu **Outils** → **Gérer les paquets...**
3. Une fenêtre s'ouvre. Dans la barre de recherche, tape : `djitellopy`
4. Clique sur **Rechercher sur PyPI**
5. Dans la liste de résultats, clique sur `djitellopy`
6. Une description s'affiche : *"Tello drone library including support for video streaming, swarms, state packets and more"*
7. Clique sur le bouton **Installer**
8. Attends quelques secondes — Thonny télécharge et installe le module
9. Quand c'est fini, tu peux fermer la fenêtre

✅ Le module est installé pour ton Python. Tu pourras l'utiliser dans tous tes programmes.

💡 **Bonus pour ta culture** : la méthode "pro" pour installer un paquet Python s'appelle `pip` (la commande à taper dans un terminal serait `pip install djitellopy`). Thonny fait exactement ça en coulisses pour toi, mais avec un beau bouton à la place de la ligne de commande.

Étape 2 : Allumer et connecter ton drone

1. **Allume le drone** : appuie longuement sur le petit bouton latéral. Une LED s'allume.
2. **Sur ton PC**, ouvre la liste des réseaux WiFi
3. Tu devrais voir un réseau qui s'appelle `TELLO-XXXXXX` (le `XXXXXX` change selon ton drone)
4. **Connecte-toi à ce WiFi**. Pas de mot de passe — c'est ouvert.
5. Vérifie l'indicateur WiFi de ton PC : tu es bien connecté ?

⚠️ **Note importante** : quand tu es connecté au WiFi du drone, **tu n'as plus d'Internet via cette carte WiFi**. Le drone n'a pas accès au web — il crée son propre réseau local entre lui et ton PC.

Si tu as besoin de garder Internet pendant le développement (consulter de la doc, télécharger un module...), trois options :

Solution	Comment
🔵 **Tethering Bluetooth**	Active le partage de connexion Bluetooth sur ton téléphone, connecte ton PC au téléphone en BT. Le WiFi reste libre pour le drone.
🔌 **Ethernet**	Si ton PC a un port réseau (ou via un adaptateur USB-Ethernet) : branche un câble. Le WiFi gère le drone, l'Ethernet gère Internet.
📱 **Tethering USB**	Connecte ton téléphone par câble USB et active "Partage de connexion USB". Marche bien et recharge le téléphone.

Sinon, simplement : tu débranches le WiFi du Tello, tu te reconnectes à ton WiFi habituel le temps de chercher une info, puis tu reviens sur le Tello pour tester. Petit yo-yo mais ça marche.

Étape 3 : Le test ultime — "Hello Tello"

Crée un nouveau fichier dans Thonny (`hello_tello.py`) et tape ce code :

```python
from djitellopy import Tello

drone = Tello()
drone.connect()

batterie = drone.get_battery()
print("Batterie :", batterie, "%")

drone.end()
```

Décortiquons :

Ligne	Ce qu'elle fait
`from djitellopy import Tello`	Importe la classe `Tello` du module `djitellopy`
`drone = Tello()`	Crée un "objet drone" — c'est ta poignée pour parler au drone
`drone.connect()`	Établit la communication WiFi avec le drone
`drone.get_battery()`	Demande au drone son niveau de batterie
`print(...)`	Affiche le résultat dans la console
`drone.end()`	Ferme proprement la communication

Exécute ce programme (▶ **ou F5**). Si tout va bien, tu vois dans la console :

```yaml
Batterie : 87 %
```

🎉 **Ton ordinateur vient de discuter avec un drone.** Le drone n'a pas bougé, mais tu as établi une vraie connexion programmatique.

⚠️ Que faire si...

Le problème	La solution
`ModuleNotFoundError: No module named 'djitellopy'`	Tu as oublié d'installer le module. Reviens à l'étape 1.
Le programme tourne sans fin sans rien afficher	Vérifie que tu es bien connecté au WiFi du drone.
`OSError: [Errno 110] Connection timed out`	Le drone est éteint, déconnecté, ou un autre PC est dessus.
Ça marche mais batterie en-dessous de 30%	Pas grave pour ce test, mais tu ne pourras pas décoller. Charge le drone.

🛡️ Avant le Premier Vol : Les Règles d'Or

> ⚠️ *À LIRE AVANT DE FAIRE VOLER LE DRONE.*



Les non-négociables

1. **Espace dégagé** : zone de vol d'au moins 3 m × 3 m, sans personne dedans (sauf les binômes pilotes)
2. **Plafond suffisant** : au moins 2,5 m de hauteur sans lustre, ventilateur ou objet suspendu
3. **Pas de survol de personne**, jamais, même pour rigoler
4. **Batterie ≥ 30%** avant tout décollage
5. **Fenêtres fermées** dans la zone de vol (un coup de vent et le drone décolle… ailleurs)
6. **Cheveux longs attachés**, pas de pendentif qui pendouille
7. **Un seul binôme à la fois** dans la zone active. Les autres regardent depuis le périmètre.

Le killer switch

Si jamais le drone part en sucette pendant ton programme, tu peux **forcer l'arrêt** :

- Dans Thonny : appuie sur le bouton rouge ■ **Stop** (ou **Ctrl + F2**)
- Le programme s'arrête, mais le drone peut continuer à hover (rester en l'air)
- Au pire, **attrape-le doucement par-dessous** quand il hover (les hélices sont sur le dessus, le dessous est safe)
- Pour un cas d'urgence absolue, **attends qu'il atterrisse seul** quand la batterie baisse — il est programmé pour ça

💡 **Truc à connaître** : pour atterrir d'urgence depuis le code Python, on peut toujours envoyer `drone.land()`. Mais si ton programme plante, il faut un autre programme pour atterrir. C'est pourquoi on fera des codes courts et clairs.

🥉 Leçon 1 : Premiers Vols

Objectif

Faire **décoller, déplacer et atterrir** ton drone avec un programme Python simple. À la fin, ton drone exécutera une mission de quelques mouvements en autonomie.

Le squelette d'un programme Tello

Toute mission Tello suit le même schéma :

```python
from djitellopy import Tello
import time

drone = Tello()
drone.connect()

# ═══ MISSION ═══
drone.takeoff()
time.sleep(2)
# ... tes mouvements ici
drone.land()

drone.end()
```

Mémorise cette structure. C'est la base de tous tes programmes :

1. **Import** du module et de `time`
2. **Créer** l'objet drone et **se connecter**
3. **Décoller** (`takeoff`)
4. **Faire** des choses
5. **Atterrir** (`land`)
6. **Fermer** la connexion (`end`)

Premier vol : décoller, hover, atterrir

Crée un nouveau fichier `vol_1.py` :

```python
from djitellopy import Tello
import time

drone = Tello()
drone.connect()
print("Batterie :", drone.get_battery(), "%")

drone.takeoff()
time.sleep(5)
drone.land()

drone.end()
```

Avant de lancer :

- Vérifie l'espace : dégagé ?
- Vérifie la batterie : ≥ 30% ?
- Pose le drone au sol, à plat, dans la zone de vol
- Recule de 1 m
- Lance ▶

Ce qui doit se passer :

1. Le drone décolle (monte d'environ 80 cm-1 m)
2. Il reste en l'air pendant 5 secondes (il **hover** = stationne en l'air)
3. Il atterrit doucement

🎉 **Bravo, ton premier vol programmé !**

Pourquoi `time.sleep(5)` après le takeoff ?

Le décollage met environ 4 secondes au Tello. Si tu enchaînes `takeoff()` puis directement `land()`, le drone reçoit l'ordre d'atterrir alors qu'il monte encore. Résultat : comportement bizarre.

Règle pratique : laisse toujours **au moins 2-3 secondes de hover entre deux commandes** pour que le drone stabilise sa position.

Les commandes de mouvement

Maintenant qu'on sait décoller et atterrir, on va apprendre à **se déplacer**.

Commande	Effet	Unité	Limites
`drone.move_forward(50)`	Avance de 50 cm	centimètres	20 à 500
`drone.move_back(50)`	Recule de 50 cm	centimètres	20 à 500
`drone.move_left(50)`	Va à gauche de 50 cm	centimètres	20 à 500
`drone.move_right(50)`	Va à droite de 50 cm	centimètres	20 à 500
`drone.move_up(50)`	Monte de 50 cm	centimètres	20 à 500
`drone.move_down(50)`	Descend de 50 cm	centimètres	20 à 500

⚠️ **Limite importante** : la distance minimum est **20 cm**. En dessous, le drone refuse la commande. La distance maximum est **500 cm** (5 mètres).

✏️ Mini-exo : un aller-retour

Modifie `vol_1.py` pour faire un aller-retour :

1. Décollage
2. Avancer de 100 cm
3. Pause de 2 secondes
4. Reculer de 100 cm
5. Atterrissage

💡 Astuce distance : développer ton intuition

Ton SDK Python te dit `move_forward(50)`. Mais est-ce que le drone fait *vraiment* 50 cm ? Et est-ce que 50 cm dans ta tête, c'est bien 50 cm dans la réalité ?

Protocole d'expérimentation (avec ton binôme) :

1. **Mesure d'abord avec ton corps.** Quelle distance fait :
   - Ton bras tendu, doigts au bout : ≈ ..... cm
   - Une largeur d'épaules : ≈ ..... cm
   - Un grand pas : ≈ ..... cm
2. **Vérifie au mètre-ruban** ces distances corporelles. Note les vraies valeurs.
3. **Marque au sol avec du scotch** des repères tous les 50 cm sur une ligne droite (0, 50, 100, 150, 200 cm). Demande au prof si tu as besoin de scotch.
4. **Pose ton drone au repère 0**, puis programme `move_forward(100)`.
5. **Lance, mesure.** Le drone est-il pile à 100 ? Ou un peu avant ? Ou un peu après ?
6. **Note tes observations** :
   - `move_forward(100)` → drone arrive à environ ..... cm

- `move_forward(50)` → drone arrive à environ ..... cm
- `move_forward(200)` → drone arrive à environ ..... cm

💡 **Ce que tu vas découvrir** : le drone n'est pas ultra-précis. Il a une marge d'erreur de quelques centimètres, et celle-ci varie selon la batterie, le sol, l'éclairage (oui oui). **C'est normal**. C'est même un bon enseignement : programmer un système physique, ce n'est pas la même chose que programmer un système numérique pur où tout est exact.

Les rotations

Pour tourner sur place, le drone peut faire deux mouvements :

Commande	Effet	Unité
`drone.rotate_clockwise(90)`	Tourne de 90° dans le sens des aiguilles d'une montre	degrés
`drone.rotate_counter_clockwise(90)`	Tourne de 90° dans le sens inverse	degrés

💡 **Astuce angles** :

- Un quart de tour = 90°
- Un demi-tour = 180°
- Un tour complet = 360°

🥉 Défi Bronze

Programme un vol qui :

1. Décolle
2. Avance de 80 cm
3. Tourne de 90° dans le sens des aiguilles d'une montre
4. Avance de 50 cm
5. Atterrit

Au final, le drone aura dessiné un quart de "L" dans l'air. **Mesurez avec votre binôme** le point d'arrivée par rapport au point de départ. Ça correspond à vos attentes ?

🥈 Leçon 2 : Structurer ton Code avec des Variables et des Boucles

Le problème de la répétition

Imagine que tu veuilles faire **un carré** avec ton drone. Tu écrirais :

```python
drone.move_forward(80)
drone.rotate_clockwise(90)
drone.move_forward(80)
drone.rotate_clockwise(90)
drone.move_forward(80)
drone.rotate_clockwise(90)
drone.move_forward(80)
drone.rotate_clockwise(90)
```

Ça marche, mais c'est **moche** : 8 lignes pour faire ce qu'on pourrait dire en français comme *"répéter 4 fois : avance et tourne à 90°"*.

En Python, on traduit ce "répéter 4 fois" avec une boucle `for`.

La boucle `for`

```python
for i in range(4):
    drone.move_forward(80)
    drone.rotate_clockwise(90)
```

Décortiquons cette syntaxe nouvelle :

Élément	Signification
`for i in range(4):`	"Pour `i` allant de 0 à 3, fais..." (donc 4 fois)
`range(4)`	Génère les nombres 0, 1, 2, 3 (4 valeurs au total)
`:` à la fin	Marque le début d'un **bloc d'instructions**
L'indentation (4 espaces)	**Crucial** : indique ce qui est dans la boucle

⚠️ **L'indentation est sacrée en Python.** Toutes les lignes qui doivent être "dans la boucle" doivent commencer par exactement le même nombre d'espaces (Thonny le fait pour toi avec la touche Tab). Si tu oublies, Python plante. Si tu mets pas le bon nombre d'espaces, Python plante.

Le rôle de `i`

Dans `for i in range(4):`, la variable `i` prend successivement les valeurs **0, 1, 2, 3**. Tu peux l'utiliser dans la boucle :

```python
for i in range(4):
    print("Tour numéro", i)
```

Affiche :

```
Tour numéro 0
Tour numéro 1
Tour numéro 2
Tour numéro 3
```

💡 **Pourquoi commencer à 0 ?** C'est une convention de programmation, et tu vas la voir partout. Pense aux niveaux d'un immeuble : en Belgique, le rez-de-chaussée n'a pas le numéro 1, il est avant. En Python, c'est pareil : on commence à 0.

Les variables pour des valeurs réutilisables

Reprends le code du carré. Et si tu voulais essayer un carré plus grand ? Tu devrais changer le `80` à plusieurs endroits, et risquer d'en oublier un.

Mieux : on met la valeur dans une variable :

```python
cote = 100  # taille d'un côté du carré, en cm

for i in range(4):
    drone.move_forward(cote)
    drone.rotate_clockwise(90)
```

Maintenant tu changes une seule valeur et tout s'adapte. C'est plus propre, plus rapide à modifier.

✏️ Mini-exo : le triangle

Programme un **triangle équilatéral** avec ton drone :

1. Décollage + pause 2s
2. Boucle : faire 3 côtés (`move_forward` puis `rotate_clockwise`)
3. Atterrissage

💡 **Indice mathématique** : pour qu'un triangle équilatéral se referme, l'angle de rotation à chaque coin n'est PAS 60°. Réfléchis : le drone doit faire un tour complet (360°) en 3 coins. Donc à chaque coin il tourne de... ?

⚠️ **Attention** : la rotation minimum côté SDK est de 1°. Pour des angles précis, n'aie pas peur de tester.

🥈 Défi Argent : le polygone régulier

Programme un drone qui dessine un **polygone à** `n` côtés (carré, pentagone, hexagone...) où `n` est une variable du programme.

```python
nombre_cotes = 5      # change cette valeur pour tester
longueur_cote = 60    # taille de chaque côté en cm
```

Pour que ton polygone se referme, il faut calculer le bon angle. C'est de la géométrie pure : tour complet (360°) ÷ nombre de côtés.

💡 **Indice** : `angle = 360 / nombre_cotes`. Tu remarqueras qu'avec `5` côtés, l'angle n'est pas un nombre rond — c'est normal, le SDK accepte les nombres entiers seulement. Tu peux convertir avec `int(360 / nombre_cotes)`.

🏅 Leçon 3 : Du Code qui Décide

Jusqu'ici, ton drone exécute aveuglément la séquence que tu lui donnes. Pas de décision : juste fais ci, fais ça. On va maintenant lui donner la **capacité de réagir à une condition**.

La structure `if / else`

Imagine que tu veuilles **refuser le décollage** si la batterie est trop faible. En français :

> *Si la batterie est supérieure à 30 %, alors décolle. Sinon, refuse et préviens.*

En Python :

```python
batterie = drone.get_battery()

if batterie > 30:
    print("Batterie OK :", batterie, "%. Décollage !")
    drone.takeoff()
else:
    print("Batterie trop faible :", batterie, "%. Vol annulé.")
```

Élément	Signification
`if condition:`	Si la condition est vraie, exécute le bloc qui suit
`else:`	Sinon, exécute l'autre bloc
`:` à la fin	Pareil que pour `for`, marque le bloc à indenter

Les opérateurs de comparaison

Opérateur	Signification	Exemple
`==`	Égal à	`age == 18`

Opérateur	Signification	Exemple
`≠`	Différent de	`couleur ≠ "rouge"`
`>`	Supérieur à	`batterie > 30`
`<`	Inférieur à	`vitesse < 100`
`⩾`	Supérieur ou égal	`score ⩾ 10`
`⩽`	Inférieur ou égal	`temperature ⩽ 0`

⚠️ **Piège classique** : `=` (un seul) sert à **assigner** une valeur à une variable. `==` (deux) sert à **comparer**.

```python
batterie = 50          # j'assigne la valeur 50 à la variable
if batterie == 50:     # je compare la variable à 50
    print("Pile poil !")
```

✏️ Mini-exo : décollage sécurisé

Écris un programme qui :

1. Se connecte au drone
2. Lit la batterie
3. **Si** batterie > 30 % : décolle, hover 3 secondes, atterrit
4. **Sinon** : affiche un message d'avertissement et ne décolle pas
5. Ferme la connexion dans tous les cas

💡 **Astuce** : pense à ce qui doit se passer **dans tous les cas** (lecture batterie, fermeture) vs **uniquement si la batterie est OK** (le vol lui-même).

Tes propres fonctions

Tu connais déjà des fonctions : `print()`, `time.sleep()`, `drone.takeoff()`... Ce sont des "outils" qui font une action quand on les appelle.

Et si tu créais les tiens ? Tu peux définir tes propres fonctions pour regrouper plusieurs lignes sous un nom parlant.

```python
def decollage_securise(drone):
    batterie = drone.get_battery()
    if batterie > 30:
        print("Batterie OK :", batterie, "%. Décollage.")
        drone.takeoff()
        return True
    else:
        print("Batterie trop faible :", batterie, "%. Annulation.")
        return False
```

Décortiquons :

Élément	Signification
`def`	Mot-clé qui crée une nouvelle fonction
`decollage_securise`	Le nom que tu donnes à ta fonction (à toi de choisir)
`(drone)`	Les "ingrédients" que la fonction attend (ici, l'objet drone)
`:`	Début du bloc de la fonction
L'indentation	Tout ce qui est indenté est **dans** la fonction
`return True` ou `return False`	Ce que la fonction **renvoie** à l'appelant

Utiliser ta fonction

Une fois définie, tu peux l'appeler comme n'importe quelle autre fonction :

```python
from djitellopy import Tello
import time

def decollage_securise(drone):
    batterie = drone.get_battery()
    if batterie > 30:
        print("Batterie OK :", batterie, "%. Décollage.")
        drone.takeoff()
        return True
    else:
        print("Batterie trop faible. Annulation.")
        return False

# ═══ MISSION ═══
drone = Tello()
drone.connect()

if decollage_securise(drone):
    time.sleep(3)
    drone.land()

drone.end()
```

🎉 **Tu viens d'écrire un programme structuré, avec une fonction réutilisable et une condition de sécurité.** C'est exactement comme ça que sont organisés les vrais programmes en entreprise.

🥇 Défi Or : la mission carrée sécurisée

Écris un programme qui combine **tout ce qu'on a appris** :

1. Définit une fonction `decollage_securise(drone)` qui vérifie la batterie
2. Définit une fonction `dessiner_carre(drone, cote)` qui dessine un carré avec une boucle `for`
3. Dans le programme principal :
   - Se connecte au drone
   - Appelle `decollage_securise`
   - Si OK : appelle `dessiner_carre(drone, 80)`, puis atterrit
   - Ferme la connexion

Si tu y arrives, **bravo** : c'est exactement comme ça que tu structurerais un vrai programme professionnel. 🌟

💎 Leçon 4 : Défi Final — Mission Atterrissage de Précision

La mission

> ***Atterris dans la zone cible, depuis le point de départ, en un programme Python.***

Le prof a marqué au scotch dans la salle :

- **Une zone de départ** (carré au sol où on pose le drone)
- **Une zone d'atterrissage cible** (cercle ou carré quelque part dans la salle)

Les règles

- **5 minutes maximum** par binôme pour le défi
- **Un seul essai** : pas de "j'ajuste et je relance dix fois"
- Tu peux **mesurer la distance** entre les deux zones avant de coder (mètre-ruban à disposition)
- Le drone doit **atterrir dans la zone cible**, pas juste survoler

Comment t'y prendre

1. **Observe** : où est le point de départ ? Où est la cible ? Quelle direction ? Quelle distance ?
2. **Mesure** au mètre-ruban : la distance et l'orientation (en avant ? sur le côté ? en biais ?)
3. **Planifie ton vol en pseudocode** :

```
DÉBUT
    Décollage sécurisé
    Avancer de X cm (avec la marge d'erreur du drone)
    [Optionnel : tourner et avancer encore si la cible n'est pas en ligne droite]
    Atterrir
FIN
```

4. **Écris ton programme** en Python
5. **Pose le drone** sur la zone de départ
6. **Lance** et regarde

Médailles selon le résultat

Niveau	Résultat
🥉 Bronze	Le drone a décollé et atterri sans crash
🥈 Argent	Le drone a atterri **dans la zone cible**, peut-être avec un peu de marge
🥇 Or	Le drone a atterri **pile au centre** de la zone cible

Niveau	Résultat
💎 Diamant	Tout ça + ton code utilise une boucle ou une fonction propre, pas juste une liste plate de commandes

🚀 Pour Aller Plus Loin

Tu as fini avant les autres ? Voici des défis bonus à attaquer **en autonomie** (le prof passe vérifier en cours de route). Lis attentivement la doc de `djitellopy` ou demande des indices.

Défi Bonus 1 : 📸 Prendre une photo en vol

Le Tello a une caméra. On peut récupérer une image et la sauver sur le PC.

Voici un squelette qui respecte ces règles :

```python
import cv2
from djitellopy import Tello
import time

def attendre_image(frame_read, timeout=10):
    debut = time.time()

    while time.time() - debut < timeout:
        frame = frame_read.frame

        if frame is not None:
            return frame

        time.sleep(0.1)

    raise Exception("Aucune image reçue après 10 secondes.")

drone = Tello()
en_vol = False

try:
    drone.connect()

    drone.streamon()
    frame_read = drone.get_frame_read()

    print("Attente du flux vidéo...")
    attendre_image(frame_read)

    drone.takeoff()
    en_vol = True
    time.sleep(2)

    print("Capture de la photo...")
    frame = attendre_image(frame_read)

    cv2.imwrite("photo_drone.jpg", frame)
    print("Photo enregistrée : photo_drone.jpg")

finally:
    if en_vol:
        drone.land()

    drone.streamoff()
    drone.end()
```

💡 **Pour installer OpenCV** : Outils → Gérer les paquets → chercher `opencv-python` → Installer.

Ton défi : programmer un drone qui décolle, **tourne sur place de 90° entre chaque photo**, et prend **4 photos** (une dans chaque direction). À la fin, tu auras 4 fichiers `photo_1.jpg` à `photo_4.jpg`.

💡 **Indice** : utilise une boucle `for` et change le nom du fichier à chaque itération avec une f-string : `f"photo_{i}.jpg"` . Pense aussi à mettre une pause entre la rotation et la photo (sinon le drone bouge encore quand tu prends la photo).

Défi Bonus 2 : 🎥 Afficher la vidéo en direct

Affiche le flux vidéo du drone dans une fenêtre, en temps réel. Mêmes règles que pour la photo : `time.sleep(3)` après `streamon`, et `frame_read` créé une seule fois.

PYTHON

```python
import cv2
from djitellopy import Tello
import time

drone = Tello()

try:
    drone.connect()
    drone.streamon()

    # Créer le lecteur tout de suite après streamon
    frame_read = drone.get_frame_read()

    # Attendre d'avoir une première vraie image
    print("Attente du flux vidéo...")

    frame = None

    while frame is None:
        frame = frame_read.frame
        time.sleep(0.1)

    print("Flux vidéo reçu. Appuie sur q pour quitter.")

    while True:
        frame = frame_read.frame

        # Sécurité : si une frame est vide, on saute ce tour
        if frame is None:
            continue

        cv2.imshow("Tello Live", frame)

        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

finally:
    drone.streamoff()
    cv2.destroyAllWindows()
    drone.end()
```

Lance, et regarde ce que voit le drone (au sol, pour commencer). Appuie sur **Q** pour quitter.

Bonus : ajoute un `drone.takeoff()` après le `time.sleep(3)` et un `drone.land()` à la fin pour voir en direct ce que voit ton drone en vol.

⚠️ Le piège du stream qui freeze pendant les commandes

Tu vas peut-être remarquer un comportement bizarre si tu combines vidéo + commandes de mouvement :

> ***Quand tu envoies*** `drone.move_forward(50)` *ou similaire, le stream vidéo se fige pendant quelques secondes, puis reprend.*

C'est **normal** et c'est **inévitable** avec les commandes classiques. Voilà pourquoi :

Le Tello a un seul "cerveau" interne et il privilégie l'exécution de la commande sur le streaming. Quand tu appelles `move_forward(50)`, ton code Python **attend que le drone ait fini son mouvement** avant de continuer. Pendant ce temps, le drone ne s'occupe pas du flux vidéo.

Solution : utiliser `send_rc_control(...)` qui est **non-bloquant**. C'est ce qu'on utilise dans le Défi Bonus 3.

Défi Bonus 3 : 🎮 Piloter au clavier (sans freeze !)

Combine vidéo + contrôle clavier pour piloter ton drone comme un jeu vidéo. **Pour que le stream reste fluide, on va utiliser** `send_rc_control` au lieu des commandes `move_*`.

Petit cours sur `send_rc_control`

Aspect	Commandes `move_forward()` etc.	`send_rc_control(lr, fb, ud, yaw)`
Type	Bloquant (attend la fin)	Non-bloquant (continu)
Stream pendant	Se fige	Reste fluide
Mode	"va à 50 cm devant"	"joystick maintenu"
Paramètres	Distance en cm	Vitesse de -100 à 100
Pour quoi	Programmes séquentiels précis	Pilotage temps réel, jeu, drone-cam

La syntaxe : `drone.send_rc_control(left_right, forward_back, up_down, yaw)`

- `left_right` : -100 (gauche) à +100 (droite). 0 = pas de mouvement latéral.
- `forward_back` : -100 (reculer) à +100 (avancer)
- `up_down` : -100 (descendre) à +100 (monter)
- `yaw` : -100 (rotation gauche) à +100 (rotation droite)

Tant que tu ne renvoies pas `(0, 0, 0, 0)`, le drone continue sur sa lancée. C'est exactement comme un joystick.

Le code

PYTHON

```python
import cv2
from djitellopy import Tello
import time

drone = Tello()
drone.connect()

drone.streamon()
time.sleep(3)
frame_read = drone.get_frame_read()

drone.takeoff()
time.sleep(2)

speed = 50  # vitesse de pilotage (1 à 100)

print("Commandes :")
print("  Z/S    : avancer / reculer")
print("  Q/D    : gauche / droite")
print("  A/E    : rotation gauche / droite")
print("  R/F    : monter / descendre")
print("  ESPACE : atterrir et quitter")

while True:
    frame = frame_read.frame
    cv2.imshow("Tello Pilot", frame)

    # Valeurs par défaut : drone immobile
    lr, fb, ud, yaw = 0, 0, 0, 0

    key = cv2.waitKey(1) & 0xFF
    if   key == ord('z'): fb = speed
    elif key == ord('s'): fb = -speed
    elif key == ord('q'): lr = -speed
    elif key == ord('d'): lr = speed
    elif key == ord('a'): yaw = -speed
    elif key == ord('e'): yaw = speed
    elif key == ord('r'): ud = speed
    elif key == ord('f'): ud = -speed
    elif key == ord(' '):
        drone.land()
        break

    drone.send_rc_control(lr, fb, ud, yaw)

drone.streamoff()
cv2.destroyAllWindows()
drone.end()
```

Touches :

- **Z/S** : avancer / reculer
- **Q/D** : gauche / droite
- **A/E** : rotation gauche / droite
- **R/F** : monter / descendre
- **Espace** : atterrir et quitter

Pourquoi ça marche bien : à chaque tour de boucle (60 fois par seconde environ), on envoie une commande RC. Si tu n'appuies sur rien, on envoie `(0, 0, 0, 0)` et le drone hover. Si tu maintiens une touche, on envoie continuellement la vitesse — le drone bouge tant que tu maintiens. Et le stream **ne se fige pas** parce qu'on n'a plus de commandes bloquantes.

💡 **Pour aller plus loin** :

- Ajoute **Page Up / Page Down** pour monter et descendre, en complément de R/F. Les codes des touches spéciales nécessitent `cv2.waitKeyEx(...)` (au lieu de `cv2.waitKey`). Cherche dans la doc OpenCV.
- Affiche **la batterie en temps réel** sur la fenêtre vidéo avec `cv2.putText(...)`.
- Ajoute une touche **P** pour prendre une photo instantanée et la sauver.

Défi Bonus 4 : 🌀 Le flip

Le Tello sait faire des **flips** (saut périlleux). Il en consomme beaucoup de batterie (5%+) — vérifie que tu es chargé.

```python
drone.flip_left()   # flip à gauche
drone.flip_right()  # flip à droite
drone.flip_forward()  # flip avant
drone.flip_back()   # flip arrière
```

⚠️ **Sécurité** : un flip prend ~80 cm horizontalement. Espace très dégagé requis.

📚 Vocabulaire à Retenir

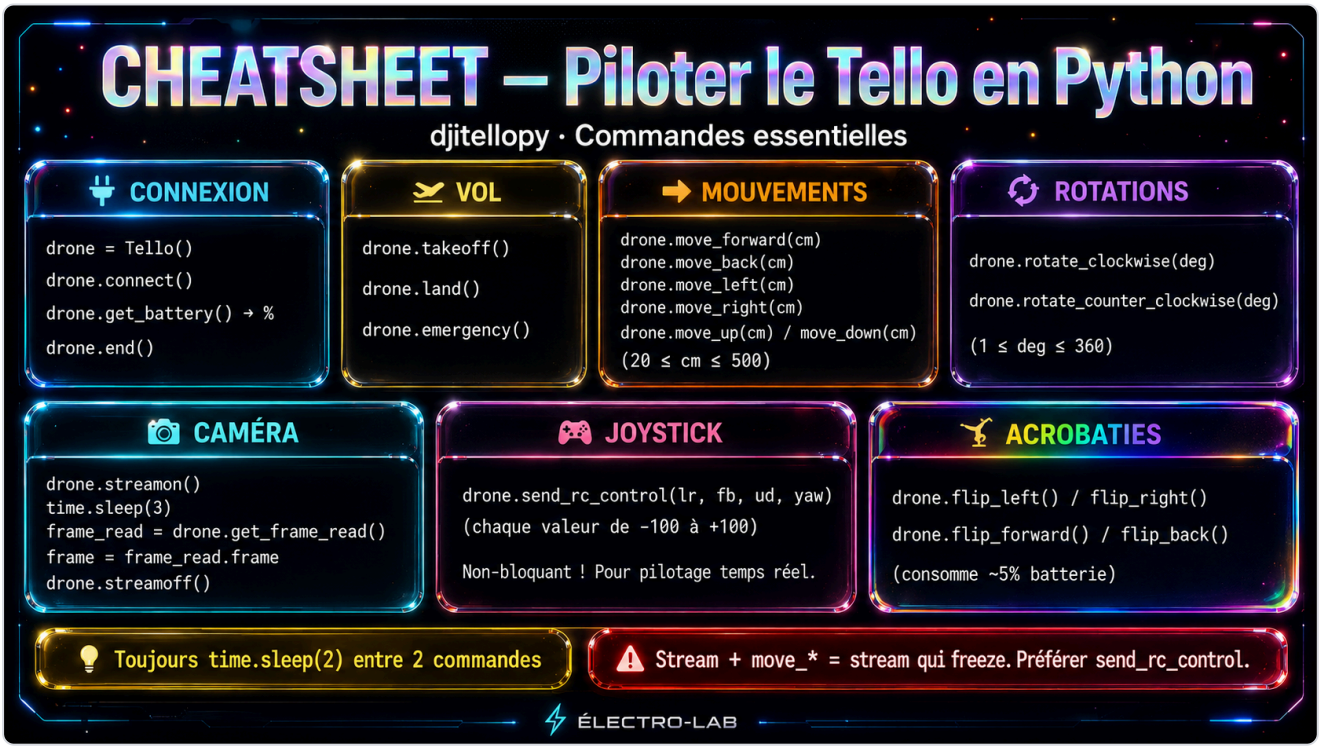
Terme	Définition
Tello	Drone DJI de 80g, contrôlable par WiFi via un PC
djitellopy	Module Python officieux mais référence pour piloter le Tello
WiFi du drone	Réseau ouvert créé par le drone (SSID `TELLO-XXXXXX`) auquel le PC se connecte
`takeoff()` / `land()`	Décollage / atterrissage

Terme	Définition
`move_forward(cm)`	Avancer de N centimètres
`rotate_clockwise(deg)`	Tourner sur place de N degrés
Hover	Stationner en l'air, en place
Boucle `for`	Répéter un bloc de code un nombre de fois donné
`if / else`	Faire un choix selon une condition
Fonction (`def` **)**	Bloc de code réutilisable, regroupé sous un nom
`streamon` / `streamoff`	Activer / couper le flux vidéo de la caméra
`send_rc_control(lr, fb, ud, yaw)`	Commande de pilotage continue, non-bloquante (style joystick)
Bloquant / Non-bloquant	Une commande **bloquante** met le programme en pause jusqu'à son achèvement. Une commande **non-bloquante** s'envoie et le code continue immédiatement.

📋 Annexe A : Catalogue de Commandes Tello Essentielles

Commande	Effet	Paramètre
`Tello()`	Crée un objet drone	—
`drone.connect()`	Établit la connexion WiFi	—
`drone.end()`	Ferme la connexion	—
`drone.get_battery()`	Renvoie le % de batterie	—
`drone.get_height()`	Renvoie la hauteur en cm	—
`drone.takeoff()`	Décollage	—
`drone.land()`	Atterrissage	—
`drone.emergency()`	⚠️ Coupe les moteurs (chute)	—
`drone.move_forward(d)`	Avance	d en cm (20-500)
`drone.move_back(d)`	Recule	idem
`drone.move_left(d)`	Gauche	idem
`drone.move_right(d)`	Droite	idem
`drone.move_up(d)`	Monte	idem

Commande	Effet	Paramètre
`drone.move_down(d)`	Descend	idem
`drone.rotate_clockwise(a)`	Tourne CW	a en degrés (1-360)
`drone.rotate_counter_clockwise(a)`	Tourne CCW	idem
`drone.flip_left()`	Flip gauche	—
`drone.flip_right()`	Flip droite	—
`drone.flip_forward()`	Flip avant	—
`drone.flip_back()`	Flip arrière	—
`drone.streamon()`	Active le flux vidéo	—
`drone.streamoff()`	Coupe le flux vidéo	—
`drone.get_frame_read().frame`	Récupère la frame courante (à créer une seule fois !)	—
`drone.send_rc_control(lr, fb, ud, yaw)`	Pilotage joystick non-bloquant	chaque valeur de -100 à 100



🆘 Annexe B : Que faire si... (FAQ Erreurs)

Erreur	Cause probable	Solution
`ModuleNotFoundError: No module named 'djitellopy'`	Module pas installé	Outils → Gérer les paquets → installer `djitellopy`
`ModuleNotFoundError: No module named 'cv2'`	OpenCV pas installé	Outils → Gérer les paquets → installer `opencv-python`
`OSError: [Errno 110] Connection timed out`	Pas connecté au WiFi du drone, ou drone éteint	Vérifier le WiFi du PC, allumer le drone
Le programme reste bloqué sans rien faire	Drone éteint pendant l'exécution, ou conflits WiFi	Stop ▶ et recommence proprement
`Aborting command "takeoff" due to error: "out of range"`	Distance hors limites (< 20 ou > 500 cm)	Ajuste les valeurs
Le drone décolle, atterrit immédiatement	Pas assez de batterie / pas assez de pause après `takeoff()`	Vérifier batterie, ajouter `time.sleep(2)` après le décollage
Le drone dérive bizarrement	Sol trop lisse / trop sombre (le capteur de positionnement souffre)	Vol sur sol contrasté (tapis, scotch au sol) ; charge la batterie
`Did not receive a state packet`	Drone connecté mais ne renvoie pas d'état	Redémarrer le drone (l'éteindre / le rallumer)
Le drone part dans la mauvaise direction	Erreur de direction dans le code	Vérifier `forward` vs `back`, `left` vs `right`, `cw` vs `ccw`
`'utf-8' codec can't decode byte 0xcc...` au tout début	Un paquet WiFi mal formé au démarrage. C'est bénin.	Ignore — le drone se reconnecte et envoie un `command: ok` juste après
`Do not have enough frames for decoding`	Tu as essayé de lire une frame trop tôt après `streamon()`	Ajoute `time.sleep(3)` après `streamon()`, et crée `frame_read = drone.get_frame_read()` UNE seule fois
`av.error.ExitError: Immediate exit requested: 'udp://...'`	Même cause que ci-dessus (le décodeur vidéo n'a pas eu assez de frames pour démarrer)	Idem : 3s d'attente après `streamon()`, et un seul `get_frame_read()`
Le stream se fige pendant `move_forward()`	C'est **normal** : les commandes de mouvement classiques sont bloquantes	Pour du pilotage en temps réel avec stream fluide, utilise `send_rc_control(...)` (voir Défi Bonus 3)

Erreur	Cause probable	Solution
`send_rc_control` mais le drone ne bouge pas	Drone pas en l'air, ou tu envoies `(0, 0, 0, 0)`	Vérifier que `takeoff()` a bien eu lieu avant ; vérifier les valeurs (-100 à +100, pas 0)

🎯 Auto-évaluation Globale

Niveau	J'ai réussi à...
🥉 Bronze	Connecter le drone, le faire décoller, voler en ligne droite, atterrir
🥈 Argent	Utiliser des variables et une boucle `for` pour dessiner un carré ou un polygone
🥇 Or	Écrire une fonction `decollage_securise` qui vérifie la batterie avec `if/else`
💎 Diamant	Réussir la mission d'atterrissage de précision avec un code structuré

Documentation

- https://github.com/damiafuentes/djitellopy
- https://djitellopy.readthedocs.io/en/latest/
- https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello SDK 2.0 User Guide.pdf