

Introduction

Pourquoi Typst ?

Le problème

Word / LibreOffice

- Éditeur d'équations limité
- Numérotations fragiles
- Mise en page compliquée
- Pas de génération avec l'IA

LaTeX

- Compilation lente
- Messages d'erreur cryptiques
- Syntaxe lourde
- Installation complexe

Note — Ces outils ont leurs forces, mais pour les documents (avec des mathématiques), il existe désormais une alternative... TYPST!

Typst : une solution moderne

Rendu
Syntaxe
Compilation
Installation

Typographie professionnelle, comparable à LaTeX
Intuitive, proche du Markdown
Instantanée, aperçu en temps réel
Éditeur en ligne gratuit ou installation en locale facile

IA

Code peut être généré par ChatGPT, Claude, Copilot (surtout Codex, Claude Code, etc.)

Comparaison : écrire une fraction

Code

LaTeX

```
\frac{-b \pm \sqrt{\Delta}}{2a}
```

Typst

```
$ \frac{-b \pm \sqrt{\Delta}}{2a} $
```

Rendu

$$\frac{-b \pm \sqrt{\Delta}}{2a}$$

Même rendu, syntaxe plus lisible.

L'argument IA

Prompt : "Génère un exercice Typst sur les équations du second degré, 3 questions avec correction."

L'IA peut générer :

- Des exercices complets avec solutions
- Des graphiques de fonctions
- Des tableaux de signes

Note — Typst étant du code texte, les LLM peuvent le produire et le modifier.

Ressources

Moodle du cours <https://edu.ge.ch/moodle/course/view.php?id=5921>
Forum du cours <https://edu.ge.ch/moodle/mod/forum/view.php?id=285602>
Documentation <https://typst.app/docs/>
Tutoriel <https://typst.app/docs/tutorial/>
Packages <https://typst.app/universe/>
Communauté [Discord Typst](#)

Passons à la pratique.

Structure de la formation

Leçon	Thème	Compétences
1	Prise en main	Compte, syntaxe de base, espacement
2	Syntaxe et maths	Formules, alignement, symboles
3	Lecture d'erreurs	Débogage, bonnes pratiques
4	Images et packages	Graphiques, taskize, zebraw
5	Personnalisation	Environnements, tableaux de signes
6	Helpers	Fonctions personnalisées, prompts IA

Résumé - Introduction : Pourquoi Typst ?

Objectif de la formation : Devenir autonome avec Typst pour créer des documents (fiches d'exercices, cours, évaluations) surtout pour les disciplines scientifiques et l'informatique.

1. Pourquoi Typst ?

Rendu élégant : documents au rendu typographique soigné, comparable à LaTeX;

Facilité d'écriture : syntaxe intuitive, proche du Markdown enrichi;

Open Source et Libre : code source ouvert, pas de licence payante;

Compilation rapide : aperçu en temps réel, pas d'attente;

Compatible IA : les documents sont du code lisible, que l'on peut relire, corriger et faire modifier ou générer par une IA.

2. Typst vs Word / LibreOffice

Critère	Word / LibreOffice	Typst
Formules mathématiques	Éditeur d'équations limité	Syntaxe native
Cohérence du style	Difficile à maintenir	Automatique via templates
Numérotation auto	Fragile, se casse	Robuste et programmable
Collaboration	Conflits de versions	Fichiers texte + Git ou typst.app
Génération par IA	Impossible	Possible
Graphiques de fonctions	Import d'images	Code intégré

Pour des documents avec des maths, Word et LibreOffice deviennent vite frustrants. Typst offre une expérience bien plus fluide.

3. Typst vs LaTeX

Critère	LaTeX	Typst
Compilation	Plusieurs secondes	Quasi-instantanée
Syntaxe	$\frac{a}{b}$	$\frac{a}{b}$
Messages d'erreurs	Peu compréhensibles	Clairs et localisés
Installation	Complexe selon l'OS	En ligne ou léger
Maturité	40+ ans, très complet	Jeune, en évolution rapide
Packages	CTAN (immense)	Universe (croissant)
Collaboration	Overleaf (payant pro)	typst.app gratuit ou installation facile en local

LaTeX reste excellent et n'est pas près de disparaître, mais pour un enseignant qui débute ou qui veut collaborer, Typst offre un rapport effort/résultat bien meilleur.

Exemple lisible : le code reste proche du rendu

Code Typst

La solution est $x = \frac{-b \text{ plus.minus } \sqrt{\text{Delta}}}{2a}$.

Rendu

La solution est $x = \frac{-b \pm \sqrt{\Delta}}{2a}$.

4. L'argument d'actualité : l'IA

Avec l'essor des LLM (ChatGPT, Claude, Copilot...), la nature « code » de Typst devient utile : on peut demander une proposition, puis la vérifier ligne par ligne. Typst reste intéressant parce que les erreurs sont visibles rapidement et que le code se corrige facilement.

L'IA peut aider à :

- corriger des erreurs dans le code ;
- générer des exercices complets avec solutions ;
- créer des représentations graphiques ;
- traduire du LaTeX ou des PDF vers Typst.

Dans cette formation en ligne, on verra brièvement comment cela fonctionne, on en reparlera plus longuement lors du cours en présentiel.

5. Structure de la formation

Leçon	Thème	Mots-clés
1	Prise en main	Compilation en ligne, syntaxe de base, espacement
2	Syntaxe et maths	Formules, alignement, symboles
3	Lecture d'erreurs	Débogage, bonnes pratiques
4	Images et packages	Graphiques, taskize, zebraw
5	Personnalisation	Environnements, tableaux de signes
6	Helpers	Fonctions personnalisées, prompts IA

6. Ressources pour aller plus loin

- **Moodle du cours** : <https://edu.ge.ch/moodle/course/view.php?id=5921>
- **Forum du cours** : <https://edu.ge.ch/moodle/mod/forum/view.php?id=285602>
- **Documentation officielle** : <https://typst.app/docs/>
- **Tutoriel interactif** : <https://typst.app/docs/tutorial/>
- **Packages Universe** : <https://typst.app/universe/>
- **Discord Typst** : Communauté active et accueillante
- **GitHub Typst** : <https://github.com/typst/typst>

Dans la leçon suivante, vous ouvrez le projet et compilez votre premier document.

Leçon 1

Prise en main

Créer son environnement

1. Créer un compte sur typst.app (gratuit)
2. Créer un nouveau projet/Dupliquer un projet existant
3. Compiler un document

Note — typst.app est l'éditeur en ligne officiel. Pas d'installation, collaboration possible, compilation en temps réel.

Structure des fichiers du cours

```
#import "../shared/lesson-layout.typ": lesson-layout
#show: lesson-layout
```

Ces lignes appliquent la mise en page commune du cours.

Attention — Pour les premières leçons, on les laisse en place sans les modifier.

Premier document

```
Code
== Mon premier document

Bonjour Typst !

== Introduction

Voici mon premier paragraphe.
```

Rendu

Mon premier document

Bonjour Typst !

Introduction

Voici mon premier paragraphe.

Commentaires

Code

```
// Commentaire sur une seule ligne

/*
Commentaire
sur plusieurs lignes
*/

Texte visible dans le document.
```

Rendu

Texte visible dans le document.

Note — Les commentaires restent dans le fichier source, mais n'apparaissent pas dans le PDF.

Texte courant et listes

Code

```
*Texte en gras*
_ Texte en italique_
#underline[Texte souligné]
`code inline`

- Liste à puces
- Deuxième élément

+ Liste numérotée
+ Deuxième étape
```

Rendu

Texte en gras

Texte en italique

Texte souligné

code inline

- Liste à puces
 - Deuxième élément
1. Liste numérotée
 2. Deuxième étape

Note — `*...*` et `_..._` sont de la syntaxe légère. Le souligné utilise une fonction : `#underline[...]`.

Réglages de base

```
#set page(margin: 2cm)
#set text(lang: "fr", size: 11pt)
```

`#set` change une règle pour la suite du document.

Niveaux de titres

Code

```
= Chapitre
== Section
=== Sous-section
==== Niveau 4
```

Rendu

Chapitre

Section

Sous-section

Niveau 4

Le nombre de `=` détermine le niveau. Avec `#set heading(numbering: "1.1.1")`, `===` peut aussi être numéroté.

Retour à la ligne vs Paragraphe

Backslash \

Même paragraphe
Lignes collées
Exemple : adresses

Ligne vide

Nouveau paragraphe
Espace vertical
Retour à la ligne

Démonstration

Code

```
Ligne 1 \  
Ligne 2 \  
Ligne 3  
  
Paragraphe 1.  
  
Paragraphe 2.
```

Rendu

Ligne 1
Ligne 2
Ligne 3

Paragraphe 1.
Paragraphe 2.

Espacement vertical : `#v()`

Code

```
Texte avant  
  
#v(1cm)  
  
Texte après  
  
#pagebreak()  
  
Nouvelle page
```

Rendu

Texte avant

Texte après

`#pagebreak()` force une nouvelle page.

Espacement horizontal : `#h()`

Code

```
Gauche #h(3cm) Droite  
  
Gauche #h(1fr) Droite
```

Rendu

Gauche Droite

Gauche Droite

Note — `#h(1fr)` utilise tout l'espace disponible, pousse vers la droite.

Alignement : `#align()`

Code

```
#align(left)[Gauche]
#align(center)[Centré]
#align(right)[Droite]

#set par(justify: true)
Texte justifié...
```

Rendu

Gauche

Centré

Droite

Texte justifié sur plusieurs lignes pour montrer l'effet du paramètre justify qui aligne le texte des deux côtés.

Attention — Pour centrer une image : `#align(center)[#image(...)]`

Unités de mesure

Unité	Exemple	Usage
cm, mm	2cm	Mesures fixes
pt	12pt	Taille de police
em	1em	Relatif à la police
%	50%	Pourcentage de la largeur
fr	1fr	Fraction de l'espace restant

Note — `em` = hauteur de la lettre « M » dans la police courante. S'adapte automatiquement si on change la taille du texte. Idéal pour les espacements.

Pièges à éviter

Erreur	Solution
Oublier <code>#</code> devant <code>v()</code>	Toujours <code>#v()</code> et <code>#h()</code>
Confondre <code>\</code> et ligne vide	Tester les deux
Trop de niveaux de titres	3 niveaux suffisent

Checklist

- ☐ Créer un document sur `typst.app`
- ☐ Écrire du gras, de l'italique, du souligné et du code inline
- ☐ Utiliser les signes égal pour les titres
- ☐ Retour à la ligne avec `backslash`
- ☐ Nouveau paragraphe avec ligne vide
- ☐ Espacement avec `#v()` et `#h()`
- ☐ Centrer avec `align(center)`
- ☐ Justifier avec `set par(justify: true)`

Leçon suivante : les mathématiques

Résumé - Prise en main

Objectifs

- Créer votre compte typst.app;
- Copier un dépôt existant;
- Compiler un document;
- Comprendre la structure d'un document;
- Maîtriser les bases de la mise en page.

1. Créer son environnement de travail

Étapes de mise en place

1. **Créer un compte** sur typst.app (gratuit, email suffit)
2. **Copier le dépôt** de formation dans votre espace
3. **Compiler** un premier document pour vérifier que tout fonctionne

Note

Éditeur en ligne officiel. Pas d'installation, collaboration possible, et compilation en temps réel.

Note RGPD : Service européen (entreprise basée en Allemagne).

Les deux premières lignes des fichiers

Les documents de formation commencent souvent par :

```
#import "../shared/lesson-layout.typ": lesson-layout
#show: lesson-layout
```

Ces lignes appliquent la mise en page commune du cours. Pour l'instant, ne les modifiez pas : elles seront expliquées plus tard avec les templates.

2. La grande question : retour à la ligne ou nouveau paragraphe ?

Retour à la ligne \	Nouveau paragraphe
Reste dans le même bloc	Crée un nouveau bloc de texte
Pas d'espacement vertical	Ajoute un espace entre les paragraphes

Exemple concret

```
// Avec \ : même paragraphe, lignes collées
Première ligne. \
Deuxième ligne. \
Troisième ligne.
```

```
// Avec ligne vide : nouveau paragraphe
Premier paragraphe avec du contenu.
```

Deuxième paragraphe, séparé visuellement.

Remarque

Attention : ici, `\` est le retour à la ligne du texte. En mode math, `\` sert surtout dans un bloc d'équations alignées, que l'on verra en leçon 2.

3. Texte courant : gras, italique, souligné, code et listes

Ces quatre syntaxes sont partout dans les documents Typst :

```
*texte important*      // gras
_mot important_        // italique
#underline[souligné]   // souligné
`#v(0.5cm)`           // code inline
```

- Élément de liste à puces
- Deuxième élément

- + Élément numéroté
- + Deuxième élément

Les listes commencent au début de la ligne. Utilisez - pour une liste à puces et + pour une liste numérotée.

Mise en forme de base

Code Typst

```
*Important* : vérifier le rendu.
_À relire_ avant envoi.
#underline[Nom et prénom] sur la copie.

- Première idée
- Deuxième idée

+ Étape 1
+ Étape 2
```

Rendu

Important : vérifier le rendu. *À relire* avant envoi. Nom et prénom sur la copie.

- Première idée
- Deuxième idée

1. Étape 1
2. Étape 2

4. Structurer avec les sections

Typst utilise le signe = pour créer des titres. Plus il y a de =, plus le niveau est profond :

```
= Chapitre (niveau 1)      // Grand titre
== Section (niveau 2)     // Sous-titre
=== Sous-section (niveau 3)
==== Niveau 4
===== Niveau 5
===== Niveau 6
```

C'est pratique! Contrairement à Word où vous cliquez sur « Titre 1 », « Titre 2 »..., ici c'est visuel et cohérent. Pas besoin de se soucier de la mise en page.

5. Contrôler l'espacement

Espacement vertical : #v()

```
#v(0.5cm)    // Ajoute 0.5 cm d'espace vertical
#v(1em)      // 1 "em" = hauteur de la police actuelle
#pagebreak() // Force une nouvelle page
```

Quand l'utiliser ?

- Entre deux environnements (on verra ce que c'est)
- Avant/après une figure importante
- Pour ajuster la mise en page d'une fiche

#v(2fr) existe aussi : fr signifie « fraction de l'espace restant ». C'est pratique pour des mises en page plus avancées, mais on n'en a pas besoin pour les premiers exercices.

Espacement horizontal : #h()

```
#h(2cm)      // Décale de 2 cm vers la droite
#h(1fr)      // Pousse vers la droite (fraction de l'espace restant)
```

Cas d'usage typiques :

- Aligner du texte manuellement
- Créer un espace entre deux éléments sur une ligne (champs de réponse)
- Pousser un élément vers la droite avec #h(1fr)

Unité	Exemple	Quand l'utiliser
cm	2cm	Mesures absolues, marges, espacements fixes
mm	5mm	Mesures précises, petits ajustements
pt	12pt	Taille de police (standard typographique)
em	1em	Relatif à la police — idéal pour les espacements
%	50%	Pourcentage de la largeur disponible
fr	1fr	Fraction de l'espace restant — très flexible

Privilégiez `em` pour les espacements dans le texte (s'adapte si on change la taille de police) et `cm` pour la mise en page générale.

Alignement : #align()

```
#align(center)[Texte centré]
#align(right)[Texte à droite]
#align(left)[Texte à gauche (défaut)]

// Centrer un bloc (graphique, image, formule...)
#align(center)[
  #image("mon-image.png", width: 50%)
]
```

Très utile pour :

- Centrer des graphiques ou images
- Aligner des blocs de contenu
- Mettre en valeur une formule importante

Centrer un contenu

Code Typst	Rendu
<code>#align(center)[Bonne formation !]</code>	Bonne formation !

Justification : #set par(justify: true) (à reconnaître)

Pour justifier le texte :

```
#set par(justify: true)
```

6. Commentaires

Les commentaires servent à laisser des notes dans le code sans les afficher dans le PDF.

```
// Commentaire sur une seule ligne

/*
Commentaire
sur plusieurs lignes
*/
```

7. Votre premier document avec des commentaires

```
// Configuration de base
#set page(margin: 2cm)           // Marges de 2 cm
#set text(lang: "fr", size: 11pt) // Français, taille 11

= Mon premier document           // Titre principal

== Introduction                   // Section

Voici mon premier paragraphe.    // Texte normal
Il continue sur cette ligne.

                                   // Ligne vide = nouveau §

Nouveau paragraphe ici.

#v(0.5cm)                        // Espace vertical

== Contenu

- Premier point                  // Liste à puces
- Deuxième point
- Troisième point

+ Première étape                 // Liste numérotée
+ Deuxième étape

#h(3cm) Texte décalé             // Décalage horizontal
```

8. Les pièges à éviter

Erreur fréquente	Solution
Oublier # devant v() ou h()	Toujours écrire #v() et #h()
Confondre \ et ligne vide	Testez les deux pour voir la différence
Espacements incohérents	Utilisez les mêmes valeurs partout

9. Checklist de fin de leçon

Avant de passer à la suite, vérifiez que vous savez :

- ☐ Créer et compiler un document sur typst.app
- ☐ Utiliser =, ==, === pour les titres
- ☐ Écrire du gras, de l'italique, du souligné, du code inline et des listes
- ☐ Faire un retour à la ligne avec \
- ☐ Créer un nouveau paragraphe avec une ligne vide
- ☐ Ajouter de l'espace avec #v() et #h()
- ☐ Choisir la bonne unité (cm, em, pt, fr)

10. Pour aller plus loin

- **Tutoriel officiel** : <https://typst.app/docs/tutorial/>
- **Référence syntaxe** : <https://typst.app/docs/reference/syntax/>
- **Guide de démarrage** : <https://typst.app/docs/guides/>

Prêt pour les mathématiques ? Direction la leçon 2 !

Leçon 1 : Prise en main de Typst — Énoncé

Objectif

Créer un document simple « Mes attentes pour cette formation » en maîtrisant :

- Les sections et sous-sections
- Le gras, l'italique, le code inline et les listes simples
- Les retours à la ligne et paragraphes
- Les espacements verticaux et horizontaux
- Le partage du projet typst.app avec le formateur

Dans typst.app, ouvrez 01-starter.typ, complétez les tâches directement dans ce fichier, compilez régulièrement et comparez avec 01-target.pdf.

Rappels de syntaxe

Sections et titres

Code	Résultat
= Titre	Titre niveau 1
== Sous-titre	Titre niveau 2
=== Sous-sous-titre	Titre niveau 3

Typst permet jusqu'à 6 niveaux avec =====.

Retour à la ligne vs nouveau paragraphe

Syntaxe	Effet
\ (backslash)	Retour à la ligne dans le même paragraphe (lignes collées)
Ligne vide	Nouveau paragraphe (espace visible entre les blocs)

Exemple :

Première ligne. \
Deuxième ligne (même paragraphe).

Nouveau paragraphe ici.

Espacements

Code	Effet
#v(0.5cm)	Espace vertical de 0.5cm
#h(2cm)	Espace horizontal de 2cm
#v(1em)	Espace vertical d'1 em (relatif à la taille du texte)

Listes

- Premier élément
- Deuxième élément
- Troisième élément

- + Première étape
- + Deuxième étape

Mise en forme inline

```
*important*      // gras
_mot_            // italique
`#v(0.5cm)`      // code inline
```

Exercice pour la section

Dans la section « Ce que j'espère apprendre » :

TODO 1 : Espace vertical

Ajouter `#v(0.5cm)` après la liste.

TODO 2 : Nouvelle section

Créer une sous-section (de niveau 2) Ce que j'aime dans l'enseignement

TODO 3 : Retour à la ligne

Écrire deux phrases séparées par `\` (même paragraphe), par exemple :

- «J'aime voir les élèves progresser.»
- «J'aime quand une explication fait clic.»

TODO 4 : Nouveau paragraphe

Après une ligne vide, écrire, par exemple : « Un nouveau paragraphe commence ici, séparé visuellement du précédent. »

TODO 5 : Espace horizontal

Créer une ligne avec un texte décalé :

Décalé : `#h(2cm)` Ce texte est décalé vers la droite.

TODO 6 : Alignement

Centrer le texte « Bonne formation ! » :

Bonne formation

TODO 7 : Liste à puces avec -

Créer une section == Mes objectifs puis une liste à puces avec -, par exemple :

- «Maîtriser la syntaxe de base»
- «Créer mes premiers documents»
- «Partager mes projets»

TODO 8 : Liste numérotée avec +

Créer une section == Je vais utiliser Typst pour puis une liste numérotée avec +, par exemple :

1. «Créer des fiches d'exercices»
2. «Rédiger mes évaluations»
3. «Préparer mes supports de cours»

TODO 9 : Partage typst.app

Partager le projet typst.app avec le formateur. Une fois les modifications faites, cliquer sur « Share » en haut à droite et entrer l'adresse email du formateur.

Checklist

- ☐ J'ai utilisé `=`, `==`, `===` pour les titres
- ☐ J'ai utilisé `\` pour un retour à la ligne
- ☐ J'ai créé un nouveau paragraphe avec une ligne vide
- ☐ J'ai utilisé `#v()` pour un espace vertical
- ☐ J'ai utilisé `#h()` pour un espace horizontal
- ☐ J'ai utilisé `#align(center)` pour centrer du contenu
- ☐ J'ai créé une liste à puces avec -

- ☐ J'ai créé une liste numérotée avec +
- ☐ Mon rendu correspond à 01-target.pdf
- ☐ Mon travail est dans 01-starter.typ
- ☐ Le projet typst.app est partagé avec le formateur

Mes attentes pour cette formation — Solution

Ce que j'espère apprendre

- Créer des fiches d'exercices professionnelles
- Écrire des formules mathématiques facilement
- Gagner du temps sur la mise en page

Ce que j'aime dans l'enseignement

J'aime voir les élèves progresser.

J'aime quand une explication fait « clic ».

Un nouveau paragraphe commence ici, séparé visuellement du précédent.

Décalé : Ce texte est décalé vers la droite.

Bonne formation !

Mes objectifs

- Maîtriser la syntaxe de base
- Créer mes premiers documents
- Partager mes projets

Je vais utiliser Typst pour

1. Créer des fiches d'exercices
2. Rédiger mes évaluations
3. Préparer mes supports de cours

Leçon 2

Syntaxe et mathématiques

Deux modes mathématiques

Inline

Dans le texte

Formule compacte

Pas d'espace : `$...$`

Display

Centré, sur sa ligne

Formule en taille normale

Espaces ou retours à la ligne

Note — Pour le display : espace après le premier `$` et avant le deuxième `$`, ou plus simplement retour à la ligne après le premier `$` et avant le deuxième `$`.

Inline vs Display

Code

```
// Inline : pas d'espace
La solution est $x = -b/(2a)$.

// Display : retours à la ligne
La solution est
$
x = -b / (2a)
$
```

Rendu

La solution est $x = -\frac{b}{2a}$.

La solution est

$$x = -\frac{b}{2a}.$$

Fractions : utiliser `\frac` d'abord

Code

```
$1/2$           // Fraction
simple
$a/b$           // Très lisible

$-b / (2a)$     // Avec
parenthèses

$frac(x + 1, x - 1)$ // Fraction
structurée
```

Rendu

$$\frac{1}{2}$$

$$\frac{a}{b}$$

$$-\frac{b}{2a}$$

$$\frac{x+1}{x-1}$$

Astuce — Utilisez `/` d'abord, gardez `frac(...)` pour les fractions longues ou imbriquées.

Fractions inline trop petites

Code

```
// Trop petit :  
La réponse est  $\frac{a}{b}$ .  
  
// Solution :  
La réponse est  $\displaystyle\left(\frac{a}{b}\right)$ .
```

Rendu

La réponse est $\frac{a}{b}$.

La réponse est $\frac{a}{b}$.

Astuce — `display()` force le rendu en taille plus grande même en mode inline.

Alignement d'équations

Code

```

$$\begin{aligned} 3x + 5 &= 14 \setminus \\ 3x &= 14 - 5 \setminus \\ 3x &= 9 \setminus \\ x &= 3 \end{aligned}$$

```

Rendu

$$\begin{aligned} 3x + 5 &= 14 \\ 3x &= 14 - 5 \\ 3x &= 9 \\ x &= 3 \end{aligned}$$

- `&` se place **devant** le symbole à aligner
- `\` termine chaque ligne **sauf la dernière**

Systèmes d'équations

Code

```

$$\begin{cases} 2x + y = 7, \\ x - y = 2, \end{cases}$$

```

Rendu

$$\begin{cases} 2x + y = 7 \\ x - y = 2 \end{cases}$$

Attention — Ne pas oublier les **virgules** entre chaque équation.

Symboles courants

`>=`

`!=`

`plus.minus`

`RR`

`in`

`>`

`≠`

`±`

`ℝ`

`∈`

`<=`

`approx`

`oo`

`NN`

`arrow`

`≤`

`≈`

`∞`

`ℕ`

`→`

Liste de tous les symboles

<https://typst.app/docs/reference/symbols/>

<https://typst.app/docs/reference/symbols/sym>

Racines et vecteurs

Code	Rendu	Usage
<code>\$sqrt(x)\$</code>	→ $\sqrt{x}$	racine carrée
<code>\$root(3, x)\$</code>	→ $\sqrt[3]{x}$	racine cubique
<code>\$vec(v)\$</code>	→ (v)	vecteur
<code>\$overline(A B)\$</code>	→ $\overline{AB}$	segment

Pour plus sur comment écrire des maths :

<https://typst.app/docs/reference/math/>

Fonction par morceaux

Code

```
$  
f(x) = cases(  
  x^2 & "si" x >= 0,  
  -x & "si" x < 0,  
)  
$
```

Rendu

$$f(x) = \begin{cases} x^2 & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

- & sépare l'expression de la condition (alignement)
- Texte entre guillemets : "si"
- Virgule après chaque cas

Espacement en mode math

Code	Rendu
<code>\$a thin b\$</code>	→ $a\,b$ (petit)
<code>\$a med b\$</code>	→ $a\,b$ (moyen)
<code>\$a quad b\$</code>	→ $a\quad b$ (grand, 1 em)

Code

```
$  
x = 3 quad "car" quad x dot x = 9  
$
```

Rendu

$$x = 3 \quad \text{car} \quad x \cdot x = 9$$

Le texte entre guillemets "car" n'est pas en italique.

Indices et exposants

Code

```
$x^2$  
$x_1$  
$x_1^2$  
$x_{(n+1)}$  
$\sum_{i=1}^n i$
```

Rendu

- x^2 exposant
- x_1 indice
- x_1^2 les deux
- x_{n+1} indice complexe
- $\sum_{i=1}^n i$

Attention — `$x_{n+1}$` donne $x_n + 1$, pas x_{n+1} — parenthèses obligatoires !

Checklist

- ☐ Savoir quand utiliser inline vs display
- ☐ Écrire des fractions lisibles
- ☐ Aligner des équations (esperluette et backslash)
- ☐ Connaître les symboles courants
- ☐ Utiliser cases pour fonctions par morceaux
- ☐ Indices complexes avec parenthèses

Leçon suivante : lecture d'erreurs

Liens utiles

Guide maths	https://typst.app/docs/guides/math/
Référence math	https://typst.app/docs/reference/math/
Symboles	https://typst.app/docs/reference/symbols/
Tous les symboles	https://typst.app/docs/reference/symbols/sym

Note — Ne mémorisez pas tous les symboles : gardez surtout le réflexe de chercher dans la documentation.

Résumé - Leçon 2 : Syntaxe et mathématiques

Objectifs

Maîtriser l'écriture des formules mathématiques !

1. Les deux modes mathématiques

C'est LA distinction fondamentale à comprendre :

Mode inline $\dots$	Mode display $\dots$ ou à la ligne
Dans le flux du texte	Centré, sur sa propre ligne
Formule compacte	Formule en taille normale
Pour les petites expressions	Pour les formules importantes

La règle d'or

// INLINE : pas d'espace après/avant le \$
La solution est $x = 3$.

// DISPLAY : espace après le premier \$ et avant le deuxième
 $x = \frac{-b \pm \sqrt{\Delta}}{2a}$

// Ou plus simplement : retour à la ligne après le premier \$
// et avant le deuxième \$
$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Note

Pour obtenir une formule display, mettez un espace après le premier \$ et avant le deuxième \$, ou plus simplement un retour à la ligne après le premier \$ et avant le deuxième \$. Typst lit ces espaces ou retours à la ligne pour décider si la formule reste dans la ligne ou devient un bloc mis en évidence.

2. Écrire des fractions

Syntaxe de base

```
$a/b$           // Fraction simple, très pratique
$frac(a, b)$    // Fraction a/b
$frac(x + 1, x - 1)$ // Avec des expressions
$frac(1, 2) + frac(3, 4)$ // Plusieurs fractions
```

Pour les petites fractions comme $\frac{1}{2}$ ou $\frac{a}{b}$, la syntaxe a/b est souvent la plus naturelle. Gardez $frac(a, b)$ pour les fractions plus structurées, par exemple $frac(x + 1, x - 1)$.

Fractions : raccourci et forme complète

Code Typst

Une demi-heure : $\frac{1}{2}$ h.

Une fraction structurée : $frac(x + 1, x - 1)$.

Rendu

Une demi-heure : $\frac{1}{2}$ h.

Une fraction structurée : $\frac{x+1}{x-1}$.

Le problème des fractions inline

Dans le texte, $frac(a, b)$ donne une fraction minuscule : $\frac{a}{b}$

Solution : Forcer le mode display avec `display()` :

```
$display(frac(a, b))$ // Grande fraction dans le texte
```

3. Aligner des équations

Pour résoudre une équation pas à pas, on aligne sur le signe = :

```
$                                // & = point d'alignement
3x + 5 &= 14 \                  // \ = nouvelle ligne
3x &= 14 - 5 \                  // \ = nouvelle ligne
3x &= 9 \
x &= 3
$
```

Règles à retenir :

- & se place **devant** le symbole à aligner (souvent =)
- \ termine chaque ligne **sauf la dernière**
- Tout est dans un seul bloc `$ ... $`

Astuce

L'indentation n'est pas obligatoire mais rend le code bien plus lisible !

Équations alignées

Code Typst

```
$
3x + 5 &= 14 \
3x &= 9 \
x &= 3
$
```

Rendu

$$\begin{aligned} 3x + 5 &= 14 \\ 3x &= 9 \\ x &= 3 \end{aligned}$$

4. Les symboles mathématiques essentiels

Code	Rendu	Code	Rendu
<code>>=</code>	$\geq$	<code><=</code>	$\leq$
<code>!=</code>	$\neq$	<code>approx</code>	$\approx$
<code>plus.minus</code>	$\pm$	<code>dot</code>	$\cdot$
<code>oo</code>	∞	<code>sqrt(x)</code>	$\sqrt{x}$
<code>RR</code>	$\mathbb{R}$	<code>NN</code>	$\mathbb{N}$
<code>in</code>	$\in$	<code>arrow</code>	$\rightarrow$

Pour le reste, utilisez la référence officielle des symboles plutôt que de mémoriser une grande table.

5. Fonctions par morceaux avec cases

Indispensable pour les fonctions définies différemment selon les cas :

```
$
f(x) = cases(
  x^2 & "si" x >= 0,
  -x & "si" x < 0,
)
$
```

Attention aux détails :

- Virgule après chaque cas (sauf le dernier, optionnel)

- & sépare l'expression de la condition
- Le texte doit être entre guillemets : "si"

6. Systèmes d'équations

Même syntaxe que cases, parfait pour les systèmes :

```
$
cases(
  2x + y = 7,
  x - y = 2,
)
$
```

Rendu :

$$\begin{cases} 2x + y = 7 \\ x - y = 2 \end{cases}$$

7. Espacement en mode math

Ajouter des espaces

```
$
a thin b      // Petit espace
a med b       // Espace moyen
a thick b     // Grand espace
a quad b      // Très grand espace (1 em)
$
```

Du texte dans les formules

Pour ajouter du texte explicatif dans une formule :

```
$
x = 3 quad "car" quad 3 dot 3 = 9
$
```

- "texte" : texte normal (pas en italique mathématique)
- quad : grand espace avant/après le texte

8. Indices et exposants

```
$x^2$          // x au carré
$x_1$          // x indice 1
$x_1^2$        // les deux
$x_{(n+1)}$    // indice complexe(parenthèses !)
$\sum_{i=1}^n$ // somme avec bornes
```

Piège classique

$\$x_{n+1}$ donne $x_n + 1$ (le +1 n'est pas en indice). Il faut $\$x_{(n+1)}$ pour avoir x_{n+1} .

9. Les pièges à éviter

Erreur	Correction
Fraction simple trop verbeuse	Utiliser $\$a/b\$$ quand cela suffit
Fraction trop petite inline	Utiliser $\text{display}(\text{frac}(\dots))$
Oublier & pour l'alignement	Placer & devant chaque =
Oublier \ en fin de ligne	Sauf la dernière ligne !
cases sans virgules	Virgule entre chaque cas

Erreur	Correction
Texte sans guillemets	"texte" dans les maths
* au lieu de dot	* donne $a * b$, dot donne $a \cdot b$

10. Récapitulatif : la fiche réflexe

// Formule inline (dans le texte)

La solution est $x = 3$.

// Formule display (centrée, importante)

```
$
x = \frac{-b \pm \sqrt{\Delta}}{2a}
$
```

// Fraction simple

La moitié s'écrit souvent $\frac{1}{2}$.

// Équations alignées

```
$
2x + 3 = 11 \
2x = 8 \
x = 4
$
```

// Fonction par morceaux

```
$
f(x) = \begin{cases}
x^2 & \text{si } x \geq 0, \\
-x & \text{sinon,} \\
\end{cases}
$
```

// Système d'équations

```
$
\begin{cases}
2x + y = 5, \\
x - y = 1, \\
\end{cases}
$
```

11. Checklist de fin de leçon

- ☐ Je sais quand utiliser inline vs display
- ☐ Je peux écrire des fractions lisibles
- ☐ Je maîtrise l'alignement avec & et \
- ☐ Je connais les symboles courants ($\geq$, $\leq$, $\in$, $\mathbb{R}$...)
- ☐ Je sais utiliser cases pour les fonctions par morceaux
- ☐ Je peux mettre du texte dans mes formules

12. Pour aller plus loin

- **Guide maths Typst** : <https://typst.app/docs/guides/math/>
- **Référence symboles** : <https://typst.app/docs/reference/symbols/>
- **Calculatrice symbolique** : Testez vos formules dans l'éditeur !

Les maths, c'est fait ! Passons à la lecture d'erreurs en leçon 3.

Leçon 2 : Syntaxe et mathématiques — Énoncé

Objectif

Créer une fiche d'exercices sur les équations du premier degré.

Dans typst.app, ouvrez 02-starter.typ, complétez le code directement dans ce fichier, puis comparez avec 02-target.pdf.

Le travail se fait dans le fichier starter : complétez les zones TODO dans les environnements #exo(...) indiqués.

Rappels de syntaxe

Formules inline vs display

- **Inline** : $a x + b = 0$ → dans le texte
- **Display** : $a x + b = 0$ → centrée

Pour une formule display, écrivez avec un espace après le premier \$ et avant le deuxième \$, ou plus simplement avec un retour à la ligne après le premier \$ et avant le deuxième \$.

Fractions et alignement

```
$frac(a, b)$           // fraction

$
  3x + 5 &= 14 \         // alignement avec &
  3x &= 9                // retour ligne avec \
$
```

Symboles

$\geq, \leq, \neq, \infty, \mathbb{R}, \in$

Exercices

Rappel de cours

Exercice 1 Écrire la définition d'une équation du premier degré avec la forme $ax + b = 0$, puis écrire la formule de la solution en mode display.

Résolution simple

Exercice 2 Écrire l'énoncé « Résoudre $3x - 9 = 0$ » puis la résolution avec des équations alignées.

Avec des fractions

Exercice 3 Écrire et résoudre l'équation suivante avec des fractions : $\frac{x}{2} + \frac{1}{3} = \frac{5}{6}$.

Inéquation

Exercice 4 Écrire et résoudre l'inéquation suivante : $2x + 1 \geq 7$.

Fonction affine par morceaux

Exercice 5 Écrire la fonction définie par morceaux suivante avec cases(), puis rédiger le calcul de $g(-3)$ et de $g(4)$:

$$g(x) = \begin{cases} 2x & \text{si } x \geq 0 \\ x^2 & \text{si } x < 0 \end{cases}$$

Système d'équations

Exercice 6 Écrire puis résoudre le système d'équations suivant avec une suite de `&cases()` dans la résolution séparés par des `\` afin d'aligner les systèmes entre-eux:

$$\begin{cases} x + y = 5 \\ x - y = 1 \end{cases}$$

Checklist

- ☐ Formules inline sans espace après `$`
- ☐ Formules display avec espaces ou retours à la ligne
- ☐ Alignement avec `&=` et `\`
- ☐ Fractions avec `frac()`
- ☐ Systèmes avec `cases()`

Équations du premier degré

Rappel de cours

Exercice 1 Écrire la définition d'une équation du premier degré avec la forme $ax + b = 0$, puis écrire la formule de la solution en mode display.

Solution 1 **Définition :** Une équation du premier degré à une inconnue est une équation de la forme $ax + b = 0$, où a et b sont des nombres réels avec $a \neq 0$.

La solution générale est :

$$x = -\frac{b}{a}$$

Donc $S = \{-\frac{b}{a}\}$.

Résolution simple

Exercice 2 Écrire l'énoncé « Résoudre $3x - 9 = 0$ » puis la résolution avec des équations alignées.

Solution 2 Résoudre $3x - 9 = 0$.

$$3x - 9 = 0$$

$$3x = 9$$

$$x = \frac{9}{3}$$

$$x = 3$$

La solution est $S = \{3\}$.

Avec des fractions

Exercice 3 Écrire et résoudre l'équation suivante avec des fractions : $\frac{x}{2} + \frac{1}{3} = \frac{5}{6}$.

Solution 3 Résoudre $\frac{x}{2} + \frac{1}{3} = \frac{5}{6}$.

$$\frac{x}{2} + \frac{1}{3} = \frac{5}{6}$$

$$\frac{x}{2} = \frac{5}{6} - \frac{1}{3}$$

$$\frac{x}{2} = \frac{5}{6} - \frac{2}{6}$$

$$\frac{x}{2} = \frac{3}{6}$$

$$\frac{x}{2} = \frac{1}{2}$$

$$x = 1$$

La solution est $S = \{1\}$.

Inéquation

Exercice 4 Écrire et résoudre l'inéquation suivante : $2x + 1 \geq 7$.

Solution 4

$$2x + 1 \geq 7$$

$$2x \geq 6$$

$$x \geq 3$$

L'ensemble des solutions est $S = [3; +\infty[$.

On peut aussi écrire $S = \{x \in \mathbb{R} \mid x \geq 3\}$.

Fonction affine par morceaux

Exercice 5 Écrire la fonction définie par morceaux suivante avec `cases()`, puis rédiger le calcul de $g(-3)$ et de $g(4)$:

$$g(x) = \begin{cases} 2x & \text{si } x \geq 0 \\ x^2 & \text{si } x < 0 \end{cases}$$

Solution 5

$$g(x) = \begin{cases} 2x & \text{si } x \geq 0 \\ x^2 & \text{si } x < 0 \end{cases}$$

Pour $x = -3$: comme $-3 < 0$, on utilise $g(x) = x^2$.

$$g(-3) = (-3)^2 = 9$$

Pour $x = 4$: comme $4 \geq 0$, on utilise $g(x) = 2x$.

$$g(4) = 2 \cdot 4 = 8$$

Système d'équations

Exercice 6 Écrire puis résoudre le système d'équations suivant avec une suite de `&cases()` dans la résolution séparés par des `\` afin d'aligner les systèmes entre-eux:

$$\begin{cases} x + y = 5 \\ x - y = 1 \end{cases}$$

Solution 6

$$\begin{cases} x + y = 5 \\ x - y = 1 \end{cases}$$

En additionnant les deux équations, puis en remplaçant x :

$$\begin{cases} (x + y) + (x - y) = 5 + 1 \\ x - y = 1 \end{cases}$$

$$\begin{cases} 2x = 6 \\ x - y = 1 \end{cases}$$

$$\begin{cases} x = 3 \\ x - y = 1 \end{cases}$$

$$\begin{cases} x = 3 \\ 3 - y = 1 \end{cases}$$

$$\begin{cases} x = 3 \\ y = 2 \end{cases}$$

La solution est $S = \{(3; 2)\}$.

Leçon 3

Lecture d'erreurs

Pourquoi cette leçon ?

Tout le monde fait des erreurs en codant.

La différence entre débutant et confirmé : **savoir les comprendre et les corriger rapidement.**

Note — Typst a un avantage majeur sur LaTeX : ses messages d'erreur sont clairs et utiles.

Anatomie d'un message d'erreur

```
error: unclosed delimiter
  └─ document.typ:15:3
15 | $ x = frac(a, b
   |   ^ unclosed delimiter
help: try closing the delimiter with `)`
```

Décryptage

Élément	Signification
---------	---------------

Ligne 1	Type d'erreur
---------	---------------

Ligne 2	Fichier, ligne, colonne
---------	-------------------------

Code	Le problème avec ^ pointant l'erreur
------	--------------------------------------

help:	Suggestion de correction
-------	--------------------------

Astuce — Lisez toujours jusqu'au `help:` — la solution est souvent là.

Erreur : Délimiteurs non fermés

```
Code
// Erreurs
$ x = 1
#align(center)[texte]
frac(a, b)
#link("texte")
```

```
Rendu
// Corrections
$ x = 1 $
#align(center)[texte]
frac(a, b)
#link("texte")
```

C'est l'erreur la plus fréquente.

Erreur : Expressions mal évaluées

Le # n'évalue qu'UN seul élément.

```
Code
#let a = 4
#let b = 5

// Erreur :
Résultat : #a + #b
// Affiche : 4 + 5
```

```
Rendu
#let a = 4
#let b = 5

// Correction :
Résultat : #(a + b)
// Affiche : 9
```

Attention — Dès qu'il y a une opération, utilisez #(expression)

Erreur : Imports mal formés

```
Code
// Erreur : manque les :
#import "@preview/pkg:0.1.0" fn
```

```
Rendu
// Correct
#import "@preview/pkg:0.1.0": fn
```

Erreur : Cases sans virgules

```
Code
// Erreur
$ f(x) = cases(
  x^2 & "si" x >= 0
  -x & "sinon"
) $
```

```
Rendu
// Correct
$ f(x) = cases(
  x^2 & "si" x >= 0,
  -x & "sinon",
) $
```

Erreur math : variables collées

En mode math, `xy` est lu comme un seul nom.

Code

```
// Rendu trompeur  
$ A = xy $
```

Rendu

```
// Correct : produit de x par y  
$ A = x y $
```

Attention — Pour multiplier deux variables, laissez un espace : `x y`, pas `xy`.

Méthode de débogage

1. **Lire** le message en entier (jusqu'au `help`)
2. **Localiser** l'erreur (aller à la ligne indiquée)
3. **Identifier** le type (délimiteur ? expression ? import ?)
4. **Corriger** une seule chose à la fois
5. **Recompiler** pour vérifier

Attention — Corrigez toujours la **première** erreur d'abord. Les suivantes disparaîtront peut-être.

Technique du commentaire

Si vous ne trouvez pas l'erreur, commentez du code suspect :

```
// Cette ligne pose peut-être problème :  
// #ma-fonction-suspecte()  
  
Le reste du document...
```

Technique «divide and conquer» : commentez la moitié, puis affinez.

Dictionnaire des erreurs

Message

unclosed delimiter

unexpected token

unknown variable

cannot add

unknown package

Signification

Oublié de fermer `$`, `[`, `(`

Caractère inattendu

Variable non définie

Opération impossible

Package inexistant

Les IA trouvent très rapidement l'erreur.

Cas typst.app fréquents

Cas	Réflexe
Package not found	Vérifier nom et version sur Universe
Import path broken	Vérifier <code>../</code> , dossier, majuscules
Function called in markup mode	Ajouter <code>#</code> devant la fonction
PDF inchangé	S'assurer de compiler le bon fichier

Fonction appelée en markup

Une fonction en mode markup doit être précédée de `#`.

Code

```
// Incorrect  
v(0.5cm)  
align(center)[Texte]
```

Rendu

```
// Correct  
#v(0.5cm)  
#align(center)[Texte]
```

Attention — Sans `#`, `v(0.5cm)` s'affiche en toutes lettres au lieu d'agir.

Erreurs dans les formules

À vérifier dans `$ ... $` :

- parenthèses de `frac(...)`, `sqrt(...)`, `cases(...)`
- virgules entre les cas de `cases`
- variables multipliées séparées par un espace : `x y`, pas `xy`
- texte entre guillemets : `"si"`
- `\` en fin de ligne dans les alignements

Checklist

Quand ça ne compile pas :

- ☐ Tous les `$` sont-ils fermés ?
- ☐ Tous les crochets sont-ils appariés ?
- ☐ Toutes les parenthèses sont-elles fermées ?
- ☐ Les imports sont-ils bien formés ?
- ☐ Les expressions utilisent-elles des parenthèses ?
- ☐ Les cases ont-ils des virgules ?

Leçon suivante : images et packages

Résumé - Leçon 3 : Lecture d'erreurs et bonnes pratiques

Objectifs

Savoir lire et corriger les erreurs de compilation. Adopter les bonnes pratiques.

1. Pourquoi cette leçon ?

Tout le monde fait des erreurs en codant. La différence entre un débutant et un utilisateur confirmé ?

La capacité à comprendre et corriger ses erreurs rapidement.

Typst a un avantage majeur sur LaTeX : ses messages d'erreur sont **clairs et utiles**.

2. Anatomie d'un message d'erreur

Quand quelque chose ne va pas, Typst affiche :

```
error: unclosed delimiter
  └─ document.typ:15:3
15 | $ x = frac(a, b
    | ^ unclosed delimiter
help: try closing the delimiter with `)``
```

Comment le lire :

- Ligne 1** error: unclosed delimiter — Le type d'erreur
- Ligne 2** document.typ:15:3 — Fichier, ligne 15, colonne 3
- Lignes 4-5** Le code problématique avec ^ pointant l'erreur
- « help »** Suggestion de correction — très utile !

Règle

Lisez TOUJOURS le message d'erreur en entier. La solution est souvent dans le « help » !

3. Les erreurs les plus fréquentes

Délimiteurs non fermés

C'est la première erreur : un \$, [, (, { ouvert mais jamais fermé.

Code erroné	Correction
\$ x = 1	\$ x = 1 \$
#align(center)[texte	#align(center)[texte]
frac(a, b	frac(a, b)
#link("texte)	#link("texte")

Expressions mal évaluées

Le # n'évalue qu'UN seul élément. Pour une expression, il faut des parenthèses :

Code	Résultat	Correction
#a + #b	Affiche « 4 + 5 »	\$(a + b) → « 9 »
#n^2	Affiche « 3^2 »	#calc.pow(n, 2) → « 9 »

Code	Résultat	Correction
#a*b	Erreur	#{a * b}

Règle

Dès qu'il y a une opération (+, -, *, /), mettez des parenthèses : #(expression)

Évaluer une expression entière

Code Typst

```
#let a = 4
#let b = 5
```

La somme vaut #(a + b).

Rendu

La somme vaut 9.

Imports mal formés

```
// ERREUR : manque les deux-points
#import "@preview/package:0.1.0" function
```

```
// CORRECT
#import "@preview/package:0.1.0": function
```

Cases sans virgules

```
// ERREUR : virgules manquantes
```

```
$ f(x) = cases(
  x^2 & "si" x >= 0
  -x & "sinon"
) $
```

```
// CORRECT : virgule après chaque cas
```

```
$ f(x) = cases(
  x^2 & "si" x >= 0,
  -x & "sinon",
) $
```

4. La méthode de débogage en 5 étapes

1. **Lire le message** — En entier, jusqu'au « help »
2. **Localiser l'erreur** — Aller à la ligne indiquée
3. **Identifier le type** — Délimiteur ? Expression ? Import ?
4. **Appliquer la correction** — Une seule à la fois !
5. **Recompiler** — Vérifier que c'est résolu

Important

Une erreur peut en cacher d'autres. Corrigez TOUJOURS la première erreur d'abord, puis recompilez. Les erreurs suivantes disparaîtront peut-être !

5. Outils de débogage

Commenter du code suspect

```
// Cette ligne pose peut-être problème
// #ma-fonction-suspecte()
```

```
// Test sans cette ligne
Le reste du document...
```

Technique du «divide and conquer»

Si vous ne trouvez pas l'erreur :

1. Commentez la moitié du document
2. Si ça compile → l'erreur est dans la partie commentée
3. Sinon → elle est dans la partie visible
4. Répétez en divisant par deux

6. Les bonnes pratiques

Avant de coder

- ☐ Le document compile avant mes modifications
- ☐ J'ai lu les instructions/commentaires
- ☐ J'ai une copie de sauvegarde (ou Git, vaste sujet si vous avez la motivation!)

Pendant le codage

- ☐ Je compile **souvent** ou utilise `typst watch` (sera traité en présentiel)
- ☐ Je corrige **une erreur à la fois**
- ☐ J'utilise des noms explicites pour mes variables
- ☐ J'indente mon code proprement pour faciliter la relecture

Après avoir codé

- ☐ Le PDF final correspond à mes attentes
- ☐ J'ai relu les formules mathématiques
- ☐ J'ai vérifié les numérotations
- ☐ J'ai comparé avec le `NN-target.pdf` si fourni

7. Les messages d'erreur courants (dictionnaire)

Message	Signification
unclosed delimiter	Vous avez oublié de fermer \$, [, (ou {
unexpected token	Caractère inattendu — vérifiez la syntaxe
unknown variable	Variable non définie — faute de frappe ?
cannot add	Opération impossible entre ces types
expected X, found Y	Mauvais type de donnée
unknown package	Package inexistant ou mal orthographié

8. Cas typst.app à connaître

Package not found

Vérifiez le nom, la version et la ligne d'import dans Typst Universe. Une ligne correcte ressemble à :

```
#import "@preview/nom-package:version": fonction
```

Import path broken

Vérifiez les chemins relatifs (`../`), le nom exact des dossiers et les majuscules/minuscules. Dans cette formation, les imports communs pointent souvent vers `shared/`.

Function called in markup mode

Si vous appelez une fonction hors du mode math, il faut `#` :

```
// Incorrect  
v(0.5cm)  
align(center)[Texte]
```

```
// Correct  
#v(0.5cm)  
#align(center)[Texte]
```

Sans le #, `v(0.5cm)` s'affiche littéralement dans le document au lieu d'insérer un espace vertical.

PDF does not update

Dans `typst.app`, vérifiez que vous compilez le bon fichier, généralement `NN-starter.typ`. Si vous modifiez `NN-solution.typ` mais exportez `NN-starter.typ`, le PDF ne correspondra pas à votre travail.

9. Spécial : erreurs dans les formules math

Les erreurs dans `$ ... $` sont parfois moins claires. Vérifiez :

- Les parenthèses de `frac(a, b)`, `sqrt(x)`, `cases(...)`
- Les expression `ab` pour le monôme `a b`
- Les virgules entre les cas de `cases`
- Les guillemets autour du texte : `"si"`, pas `si`
- Le `\` en fin de ligne pour l'alignement

10. Checklist de débogage rapide

Quand ça ne compile pas, vérifiez dans l'ordre :

- ☐ Tous les `$` sont-ils fermés ?
- ☐ Tous les `[` et `]` sont-ils appariés ?
- ☐ Toutes les parenthèses `()` sont-elles fermées ?
- ☐ Les imports ont-ils le format `@preview/nom:version` ?
- ☐ Les expressions utilisent-elles `#(...)` ?
- ☐ Les `cases` ont-ils des virgules ?
- ☐ Suis-je en train de compiler le bon fichier dans `submit/` ?

11. Pour aller plus loin

- **Messages d'erreur Typst** : <https://typst.app/docs/reference/>
- **Fiche du cours** : `resources/erreurs-frequentes.typ`
- **Discord Typst** : Posez vos questions à la communauté !
- **Aide inline** : Survolez les erreurs dans l'éditeur `typst.app`

Erreurs maîtrisées ! Place aux graphiques en leçon 4.

Exercice - Leçon 3 : Corriger un document qui ne compile pas

Objectif

Corriger un document Typst contenant **8 erreurs intentionnelles**.

Consignes

1. Ouvrir 03-starter.typ
2. Compiler et lire le message d'erreur
3. Corriger l'erreur, recompiler
4. Répéter jusqu'à compilation réussie
5. Comparer avec 03-target.pdf

Le document à corriger

Le fichier 03-starter.typ contient un document avec 8 erreurs. Pour chaque extrait ci-dessous, identifier l'erreur, corriger le code, puis recompiler.

Erreur 1 : Délimiteur \$ manquant

```
Delta = b^2 - 4 a c$
```

Erreur 2 : Fonction appelée sans

```
v(0.5cm)
```

Erreur 3 : Parenthèse manquante

```
$ x = frac(-b plus.minus sqrt(b^2 - 4 a c), 2 a $
```

Erreur 4 : Import mal formé

```
#import "@preview/simple-plot:0.9.0" plot
```

Erreur 5 : Package inexistant

```
#import "@preview/exercise-bank:0.2.6": exo
```

Erreur 6 : Bloc non fermé

```
#align(center)[  
  Le discriminant décide du nombre de solutions.
```

Erreur 7 : Cases sans virgules

```
$ cases(  
  2 & "si" Delta > 0  
  1 & "si" Delta = 0  
  0 & "si" Delta < 0  
) $
```

Erreur 8 : Variables multipliées collées

```
$ A = xy $
```

Checklist de validation

- ☐ Erreur 1 corrigée
- ☐ Erreur 2 corrigée
- ☐ Erreur 3 corrigée
- ☐ Erreur 4 corrigée
- ☐ Erreur 5 corrigée
- ☐ Erreur 6 corrigée
- ☐ Erreur 7 corrigée
- ☐ Erreur 8 corrigée
- ☐ Le document compile
- ☐ Le résultat correspond à 03-target.pdf

Leçon 3 - Fichier corrigé

Ce document contient la version corrigée du fichier de départ.

Rappel : Le discriminant

Le discriminant d'une équation $ax^2 + bx + c = 0$ est :

$$\Delta = b^2 - 4ac$$

La formule quadratique

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Graphique (préparation)

À retenir

Le discriminant décide du nombre de solutions.

Nombre de solutions

$$\text{Nombre de solutions} = \begin{cases} 2 & \text{si } \Delta > 0 \\ 1 & \text{si } \Delta = 0 \\ 0 & \text{si } \Delta < 0 \end{cases}$$

Aire d'un rectangle

L'aire d'un rectangle de côtés x et y est :

$$A = xy$$

Checklist

- ☒ Erreur 1 corrigée
- ☒ Erreur 2 corrigée
- ☒ Erreur 3 corrigée
- ☒ Erreur 4 corrigée
- ☒ Erreur 5 corrigée
- ☒ Erreur 6 corrigée
- ☒ Erreur 7 corrigée
- ☒ Erreur 8 corrigée

Leçon 4

Images et packages

Trouver un package

Catalogue officiel : typst.app/universe

Étape Action

- 1 Aller sur typst.app/universe
- 2 Chercher ou parcourir les catégories
- 3 Lire la documentation du package
- 4 Copier la ligne d'import

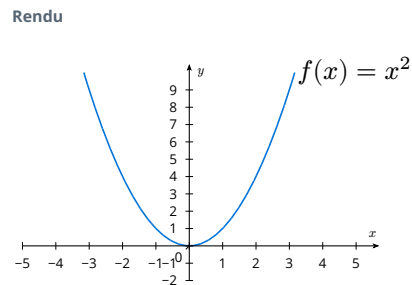
```
#import "@preview/nom-package:version": fonction
```

Graphiques avec simple-plot

<https://typst.app/universe/package/simple-plot/>

```
Code
#import "@preview/simple-plot:0.9.0":
plot

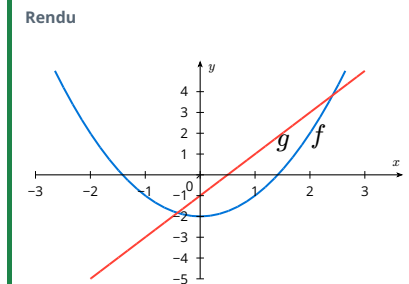
#plot(
  xmin: -5, xmax: 5.2,
  ymin: -2, ymax: 10,
  (fn: x => x*x, label: $f(x) = x^2$,
  label-pos: 0.8, label-side: "right"),
)
```



Paramètres : xmin, xmax, ymin, ymax, width, height

Plusieurs courbes

```
Code
#plot(
  xmin: -3, xmax: 3.2,
  ymin: -5, ymax: 5,
  (fn: x => x*x - 2, stroke: blue +
  1.5pt, label: $f$, label-pos: 0.8,
  label-side: "right"),
  (fn: x => 2*x - 1, stroke: red +
  1.5pt, label: $g$, label-pos: 0.7,
  label-side: "right"),
)
```



Chaque courbe : fn, stroke (couleur + épaisseur), label

Centrer les graphiques

```
Code
#align(center)[
#plot(...)
]
```

Rendu

graphique centré

Attention — Toujours centrer graphiques et images avec `#align(center)[...]`

Package taskize

<https://typst.app/universe/package/taskize/>

```
Code
#import "@preview/taskize:0.2.5": tasks

#tasks(columns: 2)[
+ Première question
+ Deuxième question
+ Troisième question
+ Quatrième question
]
```

Rendu

1) Première question	2) Deuxième question
3) Troisième question	4) Quatrième question

Répartition automatique en colonnes.

Options taskize

```
Code
#tasks(
  columns: 2,
  label-format: "a)",
  row-gutter: 1em,
)[
+ Première question
+ Deuxième question
+ Troisième question
+ Quatrième question
]
```

Rendu

a) Première question	b) Deuxième question
c) Troisième question	d) Quatrième question

`columns` répartit les questions, `label-format` change les puces, `row-gutter` règle l'espacement vertical.

Package zebraw

<https://typst.app/universe/package/zebraw/>

```
#import "@preview/zebraw:0.6.3": *
#show: zebraw

\\python
def ma_fonction(x):
  return x * x
\\
```

Coloration syntaxique automatique pour les blocs de code.

Insérer une image

```
Code
#image("chemin/image.png", width: 50%)

#align(center)[
  #image("photo.jpg", width: 70%)
]
```

Rendu

image.png

Paramètres : width, height, fit

Figure avec légende

```
Code
#figure(
  image("graphique.png", width: 70%),
  caption: [Graphique de  $f$ ],
) <fig-graphique>

Voir la @fig-graphique.
```

Rendu

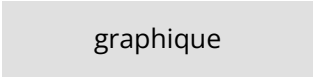
graphique

Fig. 1. – Graphique de f

Voir la Figure 1.

Tableaux simples

```
Code
#table(
  columns: 3,
  [A], [B], [C],
  [1], [2], [3],
)
```

Rendu

A	B	C
1	2	3

Tableaux avec en-tête

```
Code
#table(
  columns: (auto, 1fr, 2fr),
  table.header(
    [*N°*], [*Nom*], [*Description*]
  ),
  [1], [Item], [Texte long],
)
```

Rendu

N°	Nom	Description
1	Item	Texte long ici

auto : largeur automatique / 1fr, 2fr : proportions

Fusion de cellules

```
Code
#table(
  columns: 3,
  table.cell(colspan: 2)[Fusionné sur 2
colonnes], [C],
[A], [B], [C],
)
```

Rendu

Fusionné sur 2 colonnes		C
A	B	C

Un tableau compliqué ? Demandez à un LLM.

Grilles flexibles

```
Code
#grid(
  columns: (1fr, 2fr),
  gutter: 1em,
  [Colonne 1 (1/3)],
  [Colonne 2 (2/3)],
)
```

Rendu

Colonne 1 (1/3)	Colonne 2 (2/3)
--------------------	-----------------

grid pour la mise en page ; table pour les tableaux.

Références croisées

```
= Mon_chapitre <chap-intro>

#figure(...) <fig-1>

$ E = m c^2 $ <eq-einstein>

Voir le @chap-intro.
La @fig-1 montre le graphique.
```

Découverte de packages

1. Chercher `icons` sur Typst Universe
2. Ouvrir une page de package pertinente
3. Lire l'exemple minimal
4. Copier l'import depuis la documentation
5. Ajouter une icône devant un titre ou une remarque

Note — Même méthode pour l'exercice : trouver un package de **QR code** (par exemple `tiaoma`) et reprendre son exemple minimal.

Rendre un contenu cliquable

`#link(url)[contenu]` rend cliquable **tout** son contenu, y compris une image ou un QR code.

Code

```
// QR code statique  
#tiaoma.qrcode("https://typst.app")
```

Rendu

```
// QR code cliquable  
#link("https://typst.app") [  
  #tiaoma.qrcode("https://typst.app")  
]
```

Astuce — Le même `#link(...)` fonctionne autour d'un texte, d'une image ou d'un `#box`.

Checklist

- ☐ Importer les packages nécessaires
- ☐ Graphiques avec bornes adaptées
- ☐ Courbes de couleurs distinctes
- ☐ Questions en colonnes avec `taskize`
- ☐ Images centrées avec `align`
- ☐ Légendes avec `figure`
- ☐ QR code cliquable avec `#link`

Leçon suivante : personnalisation

Résumé - Leçon 4 : Images et packages

Objectifs

- Trouver et utiliser des packages
- Créer des graphiques avec `simple-plot`
- Structurer des questions avec `taskize`,
- Écrire du code avec `zebraw`
- Insérer des images avec légendes
- Insérer des tableaux
- Changer la mise en page avec `grid`

1. Trouver un package sur Typst Universe

Le catalogue officiel

Tous les packages Typst sont répertoriés sur :

`typst.app/universe`

Comment chercher un package

1. Aller sur `typst.app/universe`
2. Utiliser la barre de recherche ou parcourir les catégories
3. Cliquer sur un package pour voir sa documentation
4. Lire le README : objectif, exemple minimal, paramètres utiles
5. Vérifier la version actuelle, la date de mise à jour et la version minimale de Typst
6. Regarder si le package a une documentation assez claire pour être réutilisé seul
7. Copier la ligne d'import depuis la page du package

Structure d'un import

```
#import "@preview/nom-du-package:version": fonction1, fonction2
```

- `@preview` : préfixe pour les packages de la communauté
- `nom-du-package` : nom exact (sensible à la casse)
- `version` : numéro de version (ex: `0.2.0`)
- Après « : » : les fonctions à importer (* pour tout importer)

Astuce

La page du package sur Universe indique toujours la bonne syntaxe d'import. Copiez-la directement !

Mini-grille de choix

Question	Pourquoi c'est important
Le package répond-il exactement au besoin ?	Évite d'ajouter une dépendance inutile
La documentation contient-elle un exemple minimal ?	Permet de tester vite
La version est-elle récente ?	Réduit les risques d'incompatibilité
La version minimale de Typst est-elle compatible ?	Évite les erreurs de compilation

2. Package simple-plot : Graphiques de fonctions

Documentation sur Typst Universe ↗

Import et utilisation de base

```
#import "@preview/simple-plot:0.9.0": plot
```

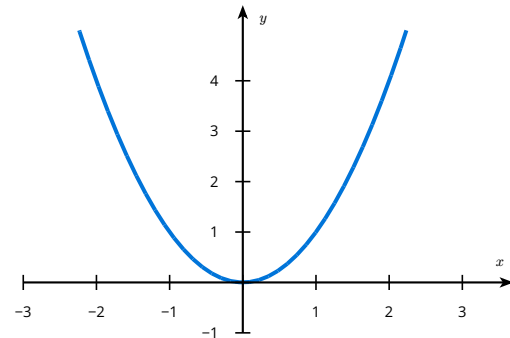
```
#plot(
  xmin: -5, xmax: 5.2,
  ymin: -2, ymax: 10,
  (fn: x => x*x, label: $f(x) = x^2$),
)
```

Graphique : code et rendu

Code Typst

```
#plot(
  xmin: -3, xmax: 3.2,
  ymin: -1, ymax: 5,
  width: 6,
  height: 4,
  (fn: x => x*x, stroke: blue + 1.5pt,
  label: $f$),
)
```

Rendu



Paramètres principaux

Paramètre	Exemple	Description
xmin, xmax	-5, 5	Bornes de l'axe des x
ymin, ymax	-2, 10	Bornes de l'axe des y
width	8cm	Largeur du graphique
height	6cm	Hauteur du graphique
show-grid	"major"	Affiche la grille principale
fn	$x \Rightarrow x*x$	Fonction à tracer (lambda)
stroke	blue + 1.5pt	Couleur et épaisseur du trait
label	f	Étiquette de la courbe

Exemple : Plusieurs courbes

```
#plot(
  xmin: -3, xmax: 3.2,
  ymin: -5, ymax: 5,
  width: 10,
  height: 7,
  (fn: x => x*x - 2, stroke: blue + 1.5pt, label: $f(x) = x^2 - 2$),
  (fn: x => 2*x - 1, stroke: red + 1.5pt, label: $g(x) = 2x - 1$),
)
```

3. Package taskize : Questions structurées

Documentation sur Typst Universe ↗

Import et utilisation

Forme générale :

```
#tasks(paramètre: valeur, autre-paramètre: valeur)[
+ Question 1
+ Question 2
]
```

Questions avec paramètres

Code Typst

```
#tasks(columns: 2, label-format: "a"),
row-gutter: 1em)[
+ Première question
+ Deuxième question
+ Troisième question
+ Quatrième question
]
```

Rendu

- | | |
|-----------------------|-----------------------|
| a) Première question | b) Deuxième question |
| c) Troisième question | d) Quatrième question |

Paramètres principaux

Paramètre	Exemple	Description
columns	2	Nombre de colonnes
label-format	"a)"	Format des étiquettes (a), 1., etc.)
row-gutter	1em	Espacement entre les lignes
above	0.5em	Espace au-dessus
below	0.5em	Espace en-dessous

4. Package zebraw : Code avec coloration

Documentation sur Typst Universe ↗

Import et utilisation

```
#import "@preview/zebraw:0.6.3": *
#show: zebraw
```

```
\\python
def ma_fonction(x):
    return x * x
\\
```

Note

#show: zebraw active la coloration pour tous les blocs de code du document. Vous pouvez aussi l'appliquer localement.

Figure avec légende et référence

Code Typst

```
#figure(
  image("../medias/figure-exemple.svg",
width: 50%),
  caption: [Figure exemple],
) <fig-exemple>

Voir @fig-exemple.
```

Rendu



Fig. 1. – Figure exemple

Voir Fig. 1.

5. Figures et images

Centrer du contenu : #align()

```
#align(center)[
  #plot(...) // ou #image(...)
]
```

Indispensable pour centrer graphiques, images, tableaux.

Insertion d'une image simple

```
#image("../medias/figure-exemple.svg", width: 50%)
```

Figure avec légende et référence

```
#figure(
  image("graphique.png", width: 70%),
  caption: [Graphique de la fonction $f$],
) <fig-graphique>
```

Comme le montre la @fig-graphique, ...

Paramètres de image

Paramètre	Exemple	Description
width	50%	Largeur (% ou unité)
height	5cm	Hauteur (optionnel)
fit	"contain"	Mode d'ajustement

6. Références croisées

Créer une étiquette

= Mon chapitre <chap-intro>

```
#figure(...) <fig-1>
```

```
$ E = m c^2 $ <eq-einstein>
```

Référencer une étiquette

Voir le @chap-intro pour l'introduction.

La @fig-1 montre le graphique.

D'après l'@eq-einstein, ...

7. Tableaux

Tableau simple

Tableau simple

Code Typst

```
#table(
  columns: 3,
  [A], [B], [C],
  [1], [2], [3],
  [4], [5], [6],
)
```

Rendu

A	B	C
1	2	3
4	5	6

Tableau avec en-tête et colonnes flexibles

Tableau avec en-tête

Code Typst

```
#table(
  columns: (auto, 1fr, 2fr),
  align: (center, left, left),
  table.header([*Col 1*], [*Col 2*],
[*Col 3*]),
  [Donnée], [Texte court], [Texte plus
long ici],
)
```

Rendu

Col 1	Col 2	Col 3
Donnée	Texte court	Texte plus long ici

Fusion de cellules

Fusion de cellules

Code Typst

```
#table(
  columns: 3,
  table.cell(colspan: 2)[Fusionné sur 2
colonnes], [C],
  [A], [B], [C],
)
```

Rendu

Fusionné sur 2 colonnes		C
A	B	C

8. Grilles flexibles

Grille flexible

Code Typst

```
#grid(
  columns: (1fr, 2fr),
  gutter: 1em,
  [Colonne 1 (1/3)],
  [Colonne 2 (2/3)],
)
```

Rendu

Colonne 1
(1/3)

Colonne 2 (2/3)

Différence avec table : grid n'a pas de bordures par défaut et est plus flexible pour la mise en page.

9. Checklist d'auto-vérification

- ☐ J'ai importé les packages nécessaires
- ☐ Mes graphiques ont des bornes adaptées
- ☐ Mes courbes ont des couleurs distinctes
- ☐ Mes questions utilisent taskize pour la mise en page
- ☐ Mes images ont des légendes avec figure
- ☐ Mes références croisées fonctionnent (@label)
- ☐ Mon code est coloré avec zebraw

10. Liens utiles

- **Typst Universe** : typst.app/universe — catalogue de tous les packages
- Package simple-plot : <https://typst.app/universe/package/simple-plot>
- Package taskize : <https://typst.app/universe/package/taskize>
- Package zebraw : <https://typst.app/universe/package/zebraw>
- Documentation images : <https://typst.app/docs/reference/visualize/image/>
- Documentation figures : <https://typst.app/docs/reference/model/figure/>

Leçon 4 : Images, graphiques et packages — Énoncé

Objectif

Créer une fiche d'exercices avec graphiques, questions numérotées, images et un package trouvé dans Typst Universe.

Dans typst.app, ouvrez 04-starter.typ, complétez le code directement dans ce fichier, puis comparez avec 04-target.pdf.

Le travail se fait dans le fichier starter : complétez les zones TODO et les environnements `#exo(...)` indiqués.

Rappels de syntaxe

simple-plot

```
#import "@preview/simple-plot:0.9.0": plot

#plot(
  xmin: -5, xmax: 5.2, ymin: -2, ymax: 10,
  width: 8, height: 6,
  (fn: x => x*x, stroke: blue + 1.5pt, label: $f$),
)
```

taskize — Questions numérotées

```
#import "@preview/taskize:0.2.5": tasks

// Basique : 1 colonne, numéros 1. 2. 3.
#tasks()[
  + Question 1
  + Question 2
]

// 2 colonnes
#tasks(columns: 2)[...]

// Lettres a) b) c)
#tasks(label-format: "a")[...]

// Espacement personnalisé
#tasks(row-gutter: 1.5em, columns: 2)[...]
```

Paramètres :

- `columns` : nombre de colonnes (1, 2, 3...)
- `label-format` : format des puces ("1.", "a"), "A.", "i.")
- `row-gutter` : espacement vertical entre questions
- `column-gutter` : espacement horizontal entre colonnes

zebraw

```
#import "@preview/zebraw:0.6.3": *
#show: zebraw
```

Lire une page Typst Universe

Quand vous découvrez un package, vérifiez :

- le nom exact et la version ;
- la date de mise à jour ;
- la version minimale de Typst ;
- l'exemple minimal ;

- les paramètres dont vous avez vraiment besoin.

Images et figures

Important : Les graphiques et images doivent être centrés avec `#align(center)[...]` :

```
#align(center)[
  #plot(...) // ou #image(...)
]

#figure(
  image("../medias/figure-exemple.svg", width: 50%),
  caption: [Légende],
) <fig-label>
```

Voir [@fig-label](#).

Exercices

Lecture graphique

Exercice 1 Dans l'environnement `solution` du starter, tracer la parabole $f(x) = x^2 - 4x + 3$ avec `simple-plot`, puis créer 4 questions avec `taskize` sur 2 colonnes.

Système d'équations

Exercice 2 Dans l'environnement `solution`, tracer les deux droites du système suivant avec `simple-plot`, en utilisant deux couleurs différentes :

$$\begin{cases} y = 2x - 1 \\ y = -x + 4 \end{cases}$$

QCM avec taskize

Exercice 3 Dans l'environnement `solution`, créer un QCM avec `taskize` pour la question « Quelle est la valeur de $f(0)$? » : 4 propositions sur 2 colonnes, avec des lettres a) b) c) d) comme puces.

Calcul mental

Exercice 4 Dans l'environnement `solution`, créer une liste de 6 calculs sur 3 colonnes avec un espacement de 1.5em entre les lignes.

Algorithme Python

Exercice 5 Afficher du code Python avec coloration syntaxique via `zebraw`.

Figure avec image

Exercice 6 Insérer l'image fournie `../medias/figure-exemple.svg` avec légende, label et référence. Quand cette version compile, vous pourrez refaire la même chose avec une image personnelle uploadée dans `medias/`.

Découverte de package : QR code

Exercice 7 Chercher un package de QR code dans Typst Universe, lire sa documentation, puis ajouter un QR code vers `https://typst.app` dans votre document.

Dans votre rendu, ajoutez aussi une phrase courte expliquant pourquoi vous avez choisi ce package.

QR code cliquable

Exercice 8 Rendre le QR code de l'exercice précédent cliquable : un clic dessus doit ouvrir <https://typst.app>. Indice : la fonction `#link(...)[...]` rend cliquable tout son contenu, y compris une image. Aller voir ici pour plus d'information : <https://typst.app/docs/reference/model/link/>.

Checklist

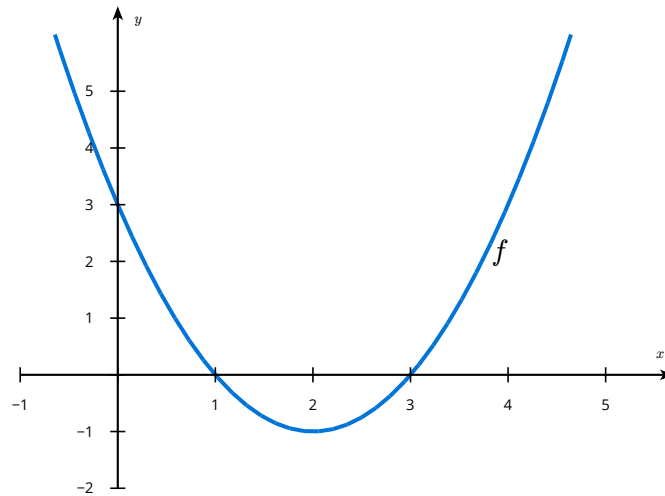
- ☐ Graphiques avec bornes adaptées
- ☐ Courbes avec couleurs distinctes et labels
- ☐ Questions avec `taskize` (colonnes, format puces)
- ☐ Code coloré avec `zbrush`
- ☐ Image avec légende et référence
- ☐ Package de QR code trouvé via Typst Universe
- ☐ QR code ajouté et documenté par une phrase
- ☐ QR code rendu cliquable avec `#link(...)`

Leçon 4 : Images, graphiques et packages — Solution

Lecture graphique

Exercice 1 Dans l'environnement `solution` du starter, tracer la parabole $f(x) = x^2 - 4x + 3$ avec `simple-plot`, puis créer 4 questions avec `taskize` sur 2 colonnes.

Solution 1



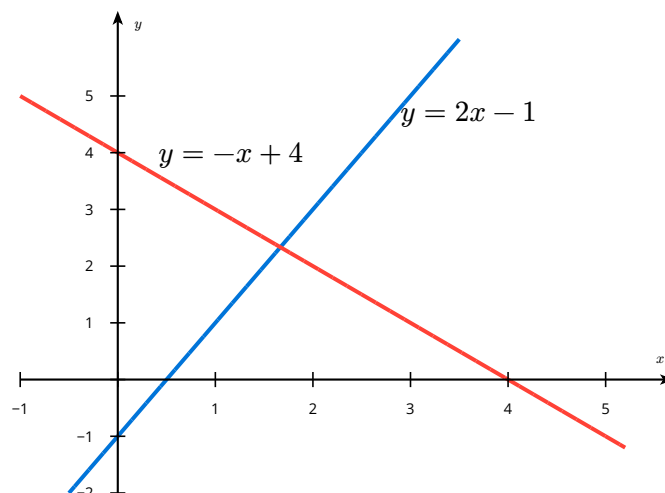
- | | |
|---|--|
| a) Déterminer graphiquement $f(0)$, $f(1)$ et $f(2)$. | b) Lire les coordonnées du sommet de la parabole. |
| c) Résoudre graphiquement $f(x) = 0$. | d) Pour quelles valeurs de x a-t-on $f(x) < 0$? |

Système d'équations

Exercice 2 Dans l'environnement `solution`, tracer les deux droites du système suivant avec `simple-plot`, en utilisant deux couleurs différentes :

$$\begin{cases} y = 2x - 1 \\ y = -x + 4 \end{cases}$$

Solution 2



QCM avec `taskize`

Exercice 3 Dans l'environnement `solution`, créer un QCM avec `taskize` pour la question « Quelle est la valeur de $f(0)$? » : 4 propositions sur 2 colonnes, avec des lettres a) b) c) d) comme puces.

Solution 3 Quelle est la valeur de $f(0)$?

a) $f(0) = 3$
c) $f(0) = -1$

b) $f(0) = 0$
d) $f(0) = 1$

Calcul mental

Exercice 4 Dans l'environnement `solution`, créer une liste de 6 calculs sur 3 colonnes avec un espacement de 1.5em entre les lignes.

Solution 4

a) $2 + 3$	b) $5 - 2$	c) $4 \cdot 3$
d) $12 \div 4$	e) 3^2	f) $\sqrt{16}$

Algorithme Python

Exercice 5 Afficher du code Python avec coloration syntaxique via `zebraw`.

Solution 5

```
1 def f(x):
2     return x**2 - 4*x + 3
3
4 print(f(0)) # Affiche 3
5 print(f(2)) # Affiche -1
```

python

Figure avec image

Exercice 6 Insérer l'image fournie `../medias/figure-exemple.svg` avec légende, label et référence. Quand cette version compile, vous pourrez refaire la même chose avec une image personnelle uploadée dans `medias/`.

Solution 6

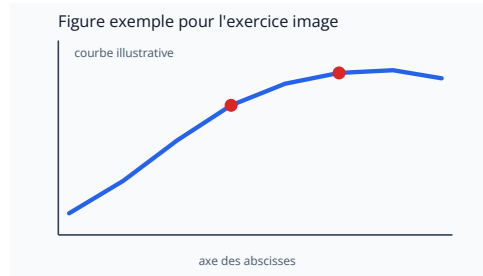


Fig. 1. – Figure exemple fournie

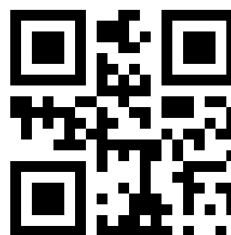
Voir la Fig. 1.

Découverte de package : QR code

Exercice 7 Chercher un package de QR code dans Typst Universe, lire sa documentation, puis ajouter un QR code vers `https://typst.app` dans votre document.

Dans votre rendu, ajoutez aussi une phrase courte expliquant pourquoi vous avez choisi ce package.

Solution 7

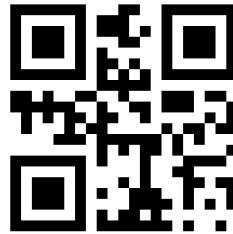


J'ai choisi tiaoma car sa page Typst Universe donne un exemple minimal direct pour générer un QR code.

QR code cliquable

Exercice 8 Rendre le QR code de l'exercice précédent cliquable : un clic dessus doit ouvrir `https://typst.app`. Indice : la fonction `#link(...)[...]` rend cliquable tout son contenu, y compris une image.

Solution 8



Leçon 5

Personnalisation

Règles set

Code

```
// Appliquer à tout le document
#set text(size: 11pt)
#set page(margin: 2cm)
#set par(justify: true)
#set text(font: police)
```

Rendu

`set` configure les éléments par défaut.

S'applique à tout ce qui suit.

On peut utiliser toute police installée sur le système. Sur `typst.app`, utilisez les polices disponibles dans le projet ou déjà installée dans l'application en ligne.

Couleurs

Code

```
// Couleurs prédéfinies
red, blue, green, gray

// Couleurs personnalisées
#let ma-couleur = rgb("#00695c")
blue.lighten(30%)
red.darken(20%)

#text(fill: blue)[Texte bleu]
#rect(fill: ma-couleur.lighten(70%),
  stroke: ma-couleur)[Encadré]
```

Rendu

Texte bleu

Encadré

Manipulation des couleurs

```
couleur.lighten(50%) // Éclaircir
couleur.darken(20%) // Assombrir
couleur.transparentize(30%) // Transparent
```



Changer la police

```
Code
// Police globale
#set text(font: "DejaVu Serif", size:
11pt)

Texte en serif

// Police locale
#text(font: "Open Sans") [
  Texte en sans-serif
]

// Avec fallback
#set text(font: ("DejaVu Serif", "Open
Sans"))
```

Rendu

Texte en serif

Texte en sans-serif

Réutiliser avec `.with()`

```
Code
#let my-tasks = tasks.with(
  columns: 2,
  label-format: "a)",
)

#my-tasks [
  + Première réponse
  + Deuxième réponse
]
```

Rendu

a) Première réponse	b) Deuxième réponse
------------------------	------------------------

`.with()` crée une version configurée d'une fonction. Cela permet d'avoir facilement, dans un même document, des listes en 1) et d'autres en a) par exemple.

Utiliser le template

```
#import "../shared/template.typ": *

#show: math-course-template.with(
  course: "IMA - EG - 2025-2026",
  license: "CC BY-NC-SA",
)
```

Le template centralise les règles `set`, `show`, les packages et les helpers.

Liens utiles : templates

- Templates sur Typst Universe :
<https://typst.app/universe/search/?kind=templates>
- `texam`, template pour évaluations :
<https://typst.app/universe/package/texam>

Liens utiles : packages

- `beautiframe` :
<https://typst.app/universe/package/beautiframe>
- `beautitled` :
<https://typst.app/universe/package/beautitled>
- `exercise-bank` :
<https://typst.app/universe/package/exercise-bank>

Titres avec beautitled

```
Code
#import "@preview/beautitled:0.2.6":
  beautitled-setup

#beautitled-setup(
  style: "titled",
  chapter-prefix: "Chapitre",
)

== Mon chapitre
== Ma section
```

Rendu

Chapitre 1

Mon chapitre

1.1 Ma section

Styles disponibles : `titled`, `schoolbook`, `modern`

Environnements beautiframe

```
Code
#import "@preview/beautiframe:0.3.0":
  definition, theorem, remark

#definition(name: "Fonction")[
  Une fonction  $f$  associe...
]

#theorem(name: "Pythagore")[
   $a^2 + b^2 = c^2$ 
]

#remark[
  Attention au cas particulier.
]
```

Rendu

Définition (Fonction)

Une fonction f associe...

Théorème (Pythagore)

$a^2 + b^2 = c^2$

Dans cette formation, `shared/template.typ` fait déjà cet import.

Configuration beautiframe

```
Code
#preset-french()
#beautiframe-setup(
  style: "modern",
  theorem-variant: "prominent",
  definition-variant: "standard",
  remark-variant: "minimal",
  accent-color: rgb("#00695c"),
)

#definition(name: "Fonction affine")[
  Une fonction de la forme...
]

#theorem(name: "Signe")[
  Si  $a > 0$ , la fonction est
  croissante.
]
```

Rendu

Définition 1 — Fonction affine

Une fonction de la forme ...

Théorème 1 — Signe

Si $a > 0$, la fonction est
croissante.

Dans cette formation, cette configuration est déjà dans `shared/template.typ`.

Tableaux de signes

```
Code
#sign-table(
  factors: (
    (expr:  $x - 2$ ,
      zeros: ((value: 2, approx: 2)),
      signs: ("-", "+")),
    (expr:  $x + 1$ ,
      zeros: ((value: -1, approx:
-1)),
      signs: ("-", "+")),
  ),
  summary-label:  $f(x)$ ,
  variation: true,
)
```

Rendu

x	-1	2
$x - 2$	-	-0+
$x + 1$	-0+	+
$f(x)$	+0-	0+

`sign-table` est fourni par le template du cours.

Exercices avec exercise-bank

```
Code
#exo(
  exercise: [
    Résoudre  $x^2 - 4 = 0$ .
  ],
  solution: [
     $x^2 = 4$  donc  $x = \pm 2$ 
  ],
)
```

Rendu

Exercice

Résoudre $x^2 - 4 = 0$.

Solution : $x = \pm 2$

Configuration des exercices

```
Code
#exo-setup(
  display: "both",
  solution-label: "Corrigé",
  exercise-label: "Question",
  badge-style: "box",
  correction-color: rgb("#1f4b99"),
)

#exo(
  exercise: [Calculer  $2 + 3$ .],
  solution: [5],
)
```

Rendu

Question 1 Calculer $2 + 3$.

Corrigé 1 5

Configurer une fois, appliquer partout.

Checklist

- ☐ Utiliser `set` pour les réglages globaux
- ☐ Utiliser `show` pour transformer l'affichage
- ☐ Personnaliser police et couleurs
- ☐ Importer le template
- ☐ Utiliser `beautitled`, `beautiframe`, `exercise-bank`

Leçon suivante : helpers

Résumé - Leçon 5 : Personnalisation

Objectifs

Utiliser le template avec beautitled, beautiframe et exercise-bank. Personnaliser l'apparence du document.

1. Structure du template

Import du template

```
#import "../shared/template.typ": *

#show: math-course-template.with(
  course: "IMA - EG - 2025-2026",
  license: "CC BY-NC-SA",
)
```

Éléments fournis par le template

Élément	Description
definition	Bloc de définition
theorem	Bloc de théorème
remark	Bloc de remarque
example	Bloc d'exemple
objectifs	Bloc d'objectifs
propriete	Bloc de propriété (français)
methode	Bloc de méthode (français)
formule	Bloc de formule (français)
exo	Exercice avec solution
plot	Graphique de fonction
tasks	Questions en colonnes

2. Environnements beautiframe

Utilisation de base

```
#definition(name: "Titre optionnel")[
  Contenu de la définition.
]

#theorem(name: "Pythagore")[
  Dans un triangle rectangle,  $a^2 + b^2 = c^2$ .
]

#remark[
  Une remarque sans titre.
]
```

Environnement de cours

Code Typst

```
#definition(name: "Discriminant") [
  Pour $a x^2 + b x + c = 0$,
  on pose $\Delta = b^2 - 4 a c$.
]
```

Rendu

Définition 1 — Discriminant

Pour $ax^2 + bx + c = 0$, on pose $\Delta = b^2 - 4ac$.

Personnalisation (à savoir situer)

Dans cette formation, `shared/template.typ` configure déjà `beautifram`. Dans un document d'exercice, utilisez d'abord `#definition`, `#theorem`, `#remark`, etc. Ne recopiez pas ce bloc dans votre fichier sauf si vous voulez vraiment modifier le template partagé.

```
#beautifram-setup(
  style: "modern",           // ou "classic", "minimal"
  theorem-variant: "prominent",
  definition-variant: "standard",
  remark-variant: "minimal",
  accent-color: rgb("#00695c"),
)
```

3. Exercices avec exercise-bank

Structure de base

```
#exo(
  exercise: [
    Énoncé de l'exercice avec $x^2 + 1 = 0$.
  ],
  solution: [
    Pas de solution réelle car $\Delta < 0$.
  ],
)
```

Configurer puis réutiliser avec .with()

Code Typst

```
#let my-tasks = tasks.with(columns: 2,
  label-format: "a)")

#my-tasks[
  + Première réponse
  + Deuxième réponse
]
```

Rendu

a) Première
réponse

b) Deuxième
réponse

Configuration globale

```
#exo-setup(
  badge-style: "box",           // Style du badge
  display: "both",              // Afficher exercice et solution
  corr-display: "correction",   // Affichage des corrections
  solution-label: "Solution",   // Étiquette
  exercise-label: "Exercice",   // Étiquette
  label-font-size: 11pt,
)
```

4. Tableaux de signes avec sign-table

Le helper sign-table est fourni par le template du cours pour générer des tableaux de signes cohérents.

```
#sign-table(
  factors: (
    (expr: $x - 2$,
      zeros: ((value: $2$, approx: 2)),
      signs: ("-", "+")),
    (expr: $x + 1$,
      zeros: ((value: $-1$, approx: -1)),
      signs: ("-", "+")),
  ),
  summary-label: $f(x)$,
  variation: true,
)
```

5. Personnalisation des titres avec beautitled

```
#beautitled-setup(
  style: "titled", // ou "schoolbook", "modern"
  chapter-prefix: "Chapitre",
  section-prefix: "Section",
)
```

6. Règles show / set (à reconnaître)

Les règles set changent un réglage pour la suite du document. Les règles show transforment automatiquement certains éléments. Pour le moment, l'objectif est surtout de les reconnaître dans le template, pas d'en écrire beaucoup.

Exemple utile et simple

```
#set page(margin: 2cm)
#set text(size: 11pt, lang: "fr")
```

Exemple avancé dans un template

```
// Tous les titres en bleu
#show heading: set text(fill: blue)
```

La syntaxe heading.where(...) existe, mais elle est plus avancée. Gardez-la pour une personnalisation de template, pas pour les premiers documents.

7. Couleurs

Couleurs prédéfinies

red, blue, green, yellow, orange, purple, black, white, gray

Couleurs personnalisées

```
rgb("#ff0000") // Rouge en hex
rgb(255, 0, 0) // Rouge en RGB
luma(50%) // Gris 50%
blue.lighten(30%) // Bleu éclairci
red.darken(20%) // Rouge assombri
```

```
// Définir sa propre couleur
#let ma-couleur = rgb("#00695c")
```

Application aux éléments

```
#text(fill: blue)[Texte bleu]
#rect(fill: ma-couleur.lighten(70%), stroke: ma-couleur)
```

8. Changer la police

Typst peut utiliser les polices installées sur le système. Sur typst.app, utilisez les polices disponibles dans le projet.

Police globale du document

```
#set text(font: "Lora", size: 11pt)
```

Police locale

```
#text(font: "Fira Sans")[Texte en sans-serif]
```

Police avec fallback

```
#set text(font: ("Lora", "serif"))
```

Si « Lora » n'est pas disponible, Typst utilise la police serif par défaut.

Polices recommandées

Type	Polices
Serif	Lora, Libertinus Serif, EB Garamond, Linux Libertine
Sans-serif	Fira Sans, Inter, Source Sans Pro, Roboto
Monospace	Fira Code, JetBrains Mono, Inconsolata

Typst.app

Sur typst.app, utilisez de préférence les polices proposées dans l'interface du projet. Si une police n'est pas disponible, Typst utilise un fallback : le document compile, mais le rendu peut changer.

9. Checklist d'auto-vérification

- ☐ J'ai importé le template correctement
- ☐ Mes définitions/théorèmes utilisent `beautiframe`
- ☐ Mes exercices ont des solutions avec `exo`
- ☐ Mes graphiques illustrent les exercices
- ☐ Le document compile sans erreur

10. Liens utiles

- Universe `beautitiled` : <https://typst.app/universe/package/beautitiled>
- Universe `beautiframe` : <https://typst.app/universe/package/beautiframe>
- Universe `exercise-bank` : <https://typst.app/universe/package/exercise-bank>
- Fonction `text` : <https://typst.app/docs/reference/text/text>
- Show rules : <https://typst.app/docs/reference/styling/>

Leçon 5 : Personnalisation — Énoncé

Objectif

Vous partez d'un document Typst déjà structuré. Votre travail consiste à le personnaliser et à compléter une fiche d'exercices.

Le but est d'utiliser des packages existants, pas de recréer un template :

- `beautitled` pour changer le style des titres ;
- `exercise-bank` pour configurer et rédiger des exercices ;
- `beautiframe` via des environnements déjà fournis par le template ;
- `taskize` via une fonction configurée avec `.with()`.

Dans `typst.app`, ouvrez `05-starter.typ`, complétez les zones `T0D0`, puis comparez avec `05-target.pdf`.

Le starter charge déjà les packages nécessaires. Vous n'avez pas à écrire les imports.

Documentation utile

- `beautitled` : <https://typst.app/universe/package/beautitled>
- `exercise-bank` : <https://typst.app/universe/package/exercise-bank>
- `beautiframe` : <https://typst.app/universe/package/beautiframe>
- `taskize` : <https://typst.app/universe/package/taskize>

Travail demandé

Chaque tâche du starter est écrite avec `#exo(...)` :

- `exercise` : [...] explique ce qu'il faut modifier ;
- `solution` : [...] contient la zone où le résultat doit être visible.

Pour les trois premiers exercices, le code à modifier se trouve dans le préambule de `05-starter.typ`, mais le résultat se vérifie dans la `solution` : de l'exercice correspondant.

1. Modifier le style des titres avec `beautitled`

Exercice 1 Au début de `05-starter.typ`, repérez le bloc `#beautitled-setup(...)`. Les modifications doivent être visibles dans la section Exercice 1 : aperçu des titres.

Modifiez le style des titres en changeant uniquement `style` et `accent-color`.

Styles possibles : `"titled"`, `"classic"`, `"modern"`, `"elegant"`, `"bold"`, `"creative"`, `"minimal"`, `"vintage"`, `"schoolbook"`, `"notes"`, `"clean"`, `"technical"`, `"academic"`, `"textbook"`, `"scholarly"`.

2. Configurer `exercise-bank`

Exercice 2 Repérez le bloc `#exo-setup(...)`. Les modifications doivent être visibles dans la section Exercice 2 : aperçu des exercices.

Modifiez la présentation des exercices : noms des badges, style du badge ou affichage des corrigés.

Valeurs utiles :

- `display` : `"ex"`, `"sol"`, `"both"` ;
- `badge-style` : `"box"`, `"circled"`, `"filled-circle"`, `"pill"`, `"tag"`, `"margin"`, `"border-accent"`, `"underline"`, `"rounded-box"`, `"header-card"` ;

- `exercise-label` et `solution-label` : textes libres.

Toutes les options sont dans le manuel : <https://typst.app/universe/package/exercise-bank>

3. Adapter la liste de questions avec `.with()`

Exercice 3 Repérez la fonction `questions` créée avec `tasks.with(...)`. Les modifications doivent être visibles dans la section Exercice 3 : aperçu des listes.

Modifiez sa configuration, puis utilisez-la dans la fiche. Par exemple, passez de a) à 1) ou changez le nombre de colonnes.

4. Configurer `beautiframe`

Exercice 4 Repérez le bloc `#beautiframe-setup(...)`. Les modifications doivent être visibles dans les blocs `important` et `definition`.

Modifiez la présentation des blocs existants. Ne créez pas de nouvel environnement.

Valeurs utiles :

- `style` : "classic", "modern", "elegant", "colorful", "boxed", "minimal", "academic", "bw", "cours" ;
- `variantes` : "prominent", "standard", "subtle", "accent", "minimal", "inline" ;
- `options de variantes` : `definition-variant`, `remark-variant`, `default-variant` ;
- `accent-color` : une couleur.

Toutes les options sont dans le manuel : <https://typst.app/universe/package/beautiframe>

5. Compléter les exercices de la fiche

Exercice 5 Dans la section Exercice 5 : exercices, complétez les trois appels `#exo(...)`.

Chaque exercice doit contenir :

- un énoncé clair dans `exercise`: [...];
- une correction dans `solution`: [...];
- au moins une utilisation de la fonction `questions`.

6. Finaliser la fiche

Exercice 6 Complétez la section Exercice 6 : méthode à retenir avec les blocs déjà présents.

Le document final doit être une fiche lisible, pas une suite de tests techniques.

Checklist

- ☐ Le fichier compile sans erreur.
- ☐ Le style `beautitiled` a été modifié.
- ☐ La configuration `exercise-bank` a été modifiée.
- ☐ La fonction `questions` utilise `.with()`.
- ☐ Les environnements existants du template sont utilisés.

☐ Les trois exercices contiennent un énoncé et une correction.

Fiche d'exercices : fonctions affines

Exercice 1 Consigne

Package utilisé : beautitled.

beautitled sert à modifier l'apparence des titres du document.

Modifier le bloc `#beautitled-setup(...)` dans le préambule, au début du fichier. Changer uniquement `style` et `accent-color`. Styles possibles : "titled", "classic", "modern", "elegant", "bold", "creative", "minimal", "vintage", "schoolbook", "notes", "clean", "technical", "academic", "textbook", "scholarly".

Résultat

Le résultat est visible sur le titre de la fiche et sur les titres ci-dessous. Ils sont affichés directement avec les fonctions de beautitled.

■ 1 Section de vérification

1.1 Sous-section de vérification

Le style utilise `style: "modern"`, une couleur personnalisée et le préfixe `Fiche`.

Exercice 2 Consigne

Configuration de exercise-bank.

Modifier le bloc `#exo-setup(...)` dans le préambule. Changer au moins deux options.

Valeurs utiles :

- `display` : "ex", "sol", "both" ;
- `badge-style` : "box", "circled", "filled-circle", "pill", "tag", "margin", "border-accent", "underline", "rounded-box", "header-card" ;
- `exercise-label` et `solution-label` : textes libres.

Toutes les options sont dans le manuel : <https://typst.app/universe/package/exercise-bank>

Résultat

L'exercice ci-dessous sert à vérifier le rendu choisi.

Question 1 Cette question montre les libellés configurés avec `#exo-setup(...)`.

Corrigé 1 Le corrigé utilise le libellé `Corrigé` et le style de badge choisi.

Exercice 3 Consigne

Liste configurée avec `.with()`.

Modifier la fonction `questions` créée avec `tasks.with(...)`. Changer `columns` OU `label-format`.

Résultat

- 1) Lire l'énoncé.
- 2) Identifier les données utiles.
- 3) Rédiger le calcul.

Exercice 4 Consigne

Package utilisé : `beautiframe`.

`beautiframe` sert à modifier l'apparence des blocs comme `important`, `definition`, `methode` OU `propriete`.

Modifier le bloc `#beautiframe-setup(...)` dans le préambule. Changer au moins deux options.

Valeurs utiles :

- `style` : "classic", "modern", "elegant", "colorful", "boxed", "minimal", "academic", "bw", "cours" ;
- `variantes` : "prominent", "standard", "subtle", "accent", "minimal", "inline" ;
- `options de variantes` : `definition-variant`, `remark-variant`, `default-variant` ;
- `accent-color` : une couleur.

Toutes les options sont dans le manuel : <https://typst.app/universe/package/beautiframe>

Résultat

Important Une fonction affine se reconnaît à son expression de la forme $ax + b$. Le coefficient a indique le sens de variation.

Définition 1 Une fonction affine est une fonction de la forme $f(x) = ax + b$. Le nombre a est la pente et le nombre b est l'ordonnée à l'origine.

Fonction affine

Exercice 5 Consigne

Exercices de la fiche.

Compléter les trois exercices ci-dessous. Ici, les appels #exo(...) sont les exercices que les élèves verraient dans le document final.

Résultat

Question 2 Soit $f(x) = 2x - 1$.

- | | |
|-------------------------|---|
| 1) Calculer $f(0)$. | 2) Calculer $f(3)$. |
| 3) Identifier la pente. | 4) Interpréter l'effet de la pente sur la droite. |

Corrigé 2 1) $f(0) = 2 \cdot 0 - 1 = -1$. 2) $f(3) = 2 \cdot 3 - 1 = 5$.

- | | |
|--------------------|--|
| 3) La pente est 2. | 4) Comme $2 > 0$, la droite monte de gauche à droite. |
|--------------------|--|

Question 3 Une fonction affine g vérifie $g(0) = -3$ et $g(2) = 1$.

- | | |
|-------------------------|--|
| 1) Déterminer la pente. | 2) En déduire une expression de $g(x)$. |
|-------------------------|--|

Corrigé 3 1) La pente vaut $\frac{1-(-3)}{2-0} = 2$. 2) Comme $g(0) = -3$, on obtient $g(x) = 2x - 3$.

Question 4 On considère $h(x) = -3x + 6$.

- | | |
|--------------------------|--------------------------|
| 1) Résoudre $h(x) = 0$. | 2) Résoudre $h(x) = 9$. |
|--------------------------|--------------------------|

Corrigé 4 1) $-3x + 6 = 0$, donc $x = 2$. 2) $-3x + 6 = 9$, donc $-3x = 3$ et $x = -1$.

Exercice 6 Consigne

Méthode à retenir.

Compléter les blocs de synthèse avec des environnements déjà fournis par le template.

Résultat

Méthode 1 | Pour étudier une fonction affine :

- | 1. repérer la pente ;
- | 2. calculer quelques images utiles ;

Propriété | Si la pente est positive, la fonction affine est croissante. S'il est négative, elle est décroissante.