

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.