BAC Pro CIEL	Python – Module 12 Gérer les erreurs avec try/except	Année 2025/2026

A. Pourquoi utiliser try ?

Quand un programme plante → il s'arrête...

Avec try → on gère les erreurs proprement

Très utile pour :

- ⇨ Fichiers inexistants
- ⇨ Saisies utilisateurs incorrects
- ⇨ Connexions réseau
- ⇨ Scripts « robustes »

B. Structure de base :

```
try :
      # code risqué
except :
      # code exécuté si erreur
```

C. Exemple simple

```
try :
      x = int(input("Entrer un nombre : "))
except :
      print("Erreur : ce n'est pas un nombre")
```

Si l'utilisateur tape « abc » → pas de crash !

D. Capturer une erreur

```
try :
      x = int(input("Entrer un nombre : "))
except ValueError :
      print("Erreur de conversion")
```

E. Les erreurs courantes à connaître

`ValueError`	Conversion impossible (int("abc"))
`FileNotFoundError`	Fichier absent
`ZeroDivisionError`	Division par zéro
`IndexError`	Index hors liste

F. Exemple avec fichier

```
try :
      with open("data.txt","r") as f :
            print(f.read())
except FileNotFoundError :
      print("Fichier introuvable")
```

G. try / except / else

```
try:
    x = int(input("Nombre : "))
except ValueError:
    print("Erreur")
else:
    print("Conversion réussie :", x)
```

else s'exécute seulement si tout va bien

H. finally

```
try:
    f = open("data.txt", "r")
except:
    print("Erreur")
finally:
    print("Fin du programme")
```

I. Récupérer le message d'erreur

```
try:
    x = int("abc")
except ValueError as e:
    print("Erreur :", e)
```

Résultat :

```
Erreur : invalid literal for int() with base 10: 'abc'
```

J. Cas concret

Vérifier si une IP entrée par l'utilisateur avec ipaddress

```
import ipaddress

try:
    ip = ipaddress.ip_address(input("IP : "))
    print("IP valide :", ip)
except ValueError:
    print("IP invalide")
```

K. Cas réseau

```python
import socket

try:
    client = socket.socket()
    client.connect(("192.168.1.10", 5000))
except ConnectionRefusedError:
    print("Connexion refusée")
except TimeoutError:
    print("Temps dépassé")
```

L. Erreurs fréquentes

Utiliser except seul (mauvaise pratique)

```
try:
    x = int(input("Nombre : "))
except:
    print("Erreur")
```

Capture toutes les erreurs et masque les vrais bugs

Oublier les deux-points :

```
try
    x = int("abc")
except ValueError
    print("Erreur")
```

```
SyntaxError: invalid syntax
```

Mettre trop de code dans le try

```
try:
    x = int(input("Nombre : "))
    print(10 / x)
    print("Fin du programme")
except ValueError:
    print("Erreur")
```

On ne sait pas quelle ligne a provoqué l'erreur

Mauvais ordre des except :

```
try:
    x = int("abc")
except:
    print("Erreur générale")
except ValueError:
    print("Erreur de conversion")
```

Le premier except capture tout. Le second ne sera jamais exécuté.

M. A retenir

try ➔ Tester un code risqué

except ➔ Gérer l'erreur

else ➔ si tout va bien

finally ➔ toujours exécuté

try contient le code susceptible de provoquer une erreur

except permet de gérer une erreur

else s'exécute seulement si aucune erreur n'a eu lieu

finally s'exécute toujours

il vaut mieux capturer une erreur précise

N. Synthèse