Comparaison : JavaScript -- ELM

JavaScript

- grande flexibilité = programmation en style impératif ou fonctionnel
- écosystème riche = davantage de bibliothèques disponibles, plus de liberté
- interopérable = facile à intégrer dans des projets web déjà existants
- exécution directe dans le navigateur (pas besoin de compilation)
- multi-paradigme = impératif, orienté objet, fonctionnel, + d'effets de bord
- typage dynamique
- pas de typage strict = sécurité faible, possibilité d'erreurs imprévisibles car peu précis
- bonne performance qui dépend de l'optimisation
- lisibilité parfois complexe sur projets volumineux
- approche rapide et intuitive avec toutes les bibliothèques (p5.js, Three.js, D3.js, Canvas API…)
- DOM modifiés via API natives ou frameworks
- gestion d'erreurs dynamiques

Elm

- + sécurisé
- langage fonctionnel fortement typé
- pas d'erreurs d'exécution = élimine plusieurs problèmes courants des langages impératifs
- paradigme entièrement fonctionnel
- bibliothèques spécifiques (elm-canvas, elm-graphics…)
- typage statique strict = prévention des erreurs avant exécution
- pas d'effets de bord = + compréhensible et facile à maintenir
- architecture Elm = structure claire, lisible et accessible
- performance optimisée grâce à compilation efficace
- écosystème plus limité (moins de bibliothèques disponibles que JS) mais bien structuré
- nécessite étape de compilation avant exécution dans le navigateur
- paradigme fonctionnel strict = prise en main moyennement accessible
- forte sécurité des types due à la compilation stricte
- robustesse, maintenabilité
- fiabilité, durabilité
- DOM manipulé de manière déclarative (architecture unidirectionnelle)
- utilise des ports pour communiquer avec du code JS
- moins flexible