| **Commande/Fonction** | **Description** | **Exemple** |
| --- | --- | --- |
| let | Déclare une variable avec un type défini ou inféré. | let x: int = 42 |
| const | Déclare une constante immuable. | const PI: float = 3.14159 |
| if ... else | Structure conditionnelle pour exécuter un code selon une condition. | if x > 0 { print("Positif") } else { print("Négatif") } |
| for | Boucle sur un intervalle ou une collection, avec options parallèles. | for i in range(0, 10).parallel() { print(i) } |
| while | Boucle conditionnelle. | while x > 0 { print(x); x -= 1 } |
| fn | Déclare une fonction avec ou sans retour. | fn ajouter(a: int, b: int) -> int { return a + b } |
| return | Renvoie une valeur depuis une fonction. | return x \* 2 |
| import | Charge un module ou une bibliothèque externe. | import math |
| export | Rend une fonction ou une variable accessible depuis d'autres modules. | export fn multiplier(a: int, b: int) -> int { return a \* b } |
| mem\_alloc / free | Alloue et libère manuellement de la mémoire. | mem\_alloc arr: hyperarray<int> = alloc(1\_000); defer free(arr) |
| defer | Planifie une action à exécuter à la sortie d’un scope. | defer print("Nettoyage") |
| match | Expression de correspondance pour simplifier les conditions complexes. | match x { 1 => print("Un"), 2 => print("Deux"), \_ => print("Autre") } |
| try ... catch | Gestion des exceptions et erreurs. | try { risky\_fn() } catch e { print("Erreur : ", e) } |
| spawn | Lancer un thread léger pour une tâche parallèle. | spawn { print("Thread lancé") } |
| sync | Synchronise plusieurs tâches ou threads. | sync { task1(); task2() } |
| async / await | Définit des fonctions asynchrones et gère leurs résultats. | let res = await async\_fn() |
| quantum | Manipule des bits quantiques pour des calculs avancés. | quantum q = quantum::init(1); q.apply(hadamard) |
| class | Définit une classe pour la programmation orientée objet. | class Point { let x: int; let y: int; fn move(dx: int, dy: int) { x += dx; y += dy } } |
| new | Instancie une classe ou une structure. | let p = new Point(10, 20) |
| meta | Permet de manipuler le code comme des objets. | meta program = load("example.eon"); program.optimize() |
| lambda | Déclare une fonction anonyme. | let carré = lambda x -> x^2 |
| module | Définit un module indépendant pour organiser le code. | module MathUtils { export fn carré(x: int) -> int { return x \* x } } |
| protocol | Définit une interface ou un contrat pour des classes/structures. | protocol Drawable { fn draw() } |
| extend | Permet d’ajouter des méthodes à une classe ou un type existant. | extend int { fn est\_pair() -> bool { return self % 2 == 0 } } |
| pipeline (` | >`) | Enchaîne des opérations sur des données. |
| debug | Imprime des informations pour le débogage. | debug("Valeur de x : ", x) |
| noop | Instruction neutre qui ne fait rien (utile pour placeholders). | noop |
| inline | Marque une fonction ou un bloc de code pour être intégré directement. | inline fn rapide(a: int, b: int) -> int { return a + b } |

**Introduction**

**Eon** est un langage de programmation conçu pour être extrêmement puissant et performant. Il permet de développer des applications très complexes tout en étant simple à comprendre et à utiliser. Ce langage est particulièrement adapté à la gestion de tâches parallèles, aux calculs scientifiques, à la manipulation de données massives et à la création d'applications web avancées.

Ce document présente les commandes de base d'Eon, ainsi que des exemples d’utilisation de ses fonctions et variables, accompagnés de leurs résultats en **HTML**.

**But de EON**

Eon vise à être un langage de programmation extrêmement puissant, capable de gérer efficacement des tâches complexes tout en maintenant une simplicité d’utilisation. Il a été conçu pour :

* Optimiser la gestion de la mémoire et des ressources.
* Exécuter des calculs scientifiques et mathématiques complexes.
* Offrir un haut niveau de parallélisme pour les applications distribuées.
* Créer des interfaces web dynamiques et interactives avec une intégration facile de bibliothèques externes.

**Commandes de base**

Voici un aperçu des commandes et concepts essentiels d'Eon.

**1. Déclaration des variables**

Les variables en Eon sont dynamiques et peuvent contenir tout type de données : nombre, chaîne de caractères, tableau, ou même fonction.

eon

Copier le code

let x = 42 // Un nombre

let nom = "Alice" // Une chaîne de caractères

let tableau = [1, 2, 3, 4] // Un tableau de nombres

**2. Fonctions**

Les fonctions en Eon sont déclarées à l’aide du mot-clé fn. Elles peuvent prendre des paramètres et retourner des valeurs.

eon

Copier le code

fn addition(a: int, b: int) -> int {

return a + b

}

**3. Boucles et itérations**

Eon supporte les boucles classiques pour effectuer des répétitions sur des données.

eon

Copier le code

for i in range(0, 10) {

print(i)

}

**4. Structures de contrôle**

Les conditions if, else, et match permettent de contrôler le flux d'exécution du programme.

eon

Copier le code

let age = 18

if age >= 18 {

print("Adulte")

} else {

print("Mineur")

}

**5. Parallélisme**

Eon permet d’exécuter plusieurs tâches en parallèle, ce qui est particulièrement utile pour les applications nécessitant un haut niveau de performance.

eon

Copier le code

let résultats = [1, 2, 3, 4, 5].parallel\_map(fn(x) -> x \* x)

**Exemples d'utilisation et résultats en HTML**

**Exemple 1 : Calcul de la somme de carrés (sans parallélisme)**

eon

Copier le code

fn somme\_des\_carres(n: int) -> int {

let somme = 0

for i in range(1, n+1) {

somme += i \* i

}

return somme

}

let resultat = somme\_des\_carres(10)

print(resultat)

**Explication :**

* Cette fonction calcule la somme des carrés des entiers de 1 à n.
* Le résultat de l'exécution sera affiché dans la console.

**Exemple 2 : Calcul parallèle de carrés (avec parallélisme)**

eon

Copier le code

let résultats = [1, 2, 3, 4, 5].parallel\_map(fn(x) -> x \* x)

print(résultats)

**Explication :**

* L’utilisation de parallel\_map permet d’effectuer les calculs de carrés en parallèle, améliorant ainsi la vitesse d'exécution.

**Exemple 3 : Calcul des nombres premiers jusqu'à un certain nombre**

eon

Copier le code

fn est\_premier(n: int) -> bool {

if n <= 1 {

return false

}

for i in range(2, n) {

if n % i == 0 {

return false

}

}

return true

}

let primes = []

for i in range(2, 50) {

if est\_premier(i) {

primes.append(i)

}

}

print(primes)

**Explication :**

* Cette fonction vérifie si un nombre est premier.
* Elle parcourt ensuite les entiers de 2 à 50 pour collecter tous les nombres premiers.

**Résultats en HTML**

Pour afficher les résultats des exemples précédents sous forme de page HTML dynamique, voici comment vous pouvez intégrer le tout :

html

Copier le code

<!DOCTYPE html>

<html lang="fr">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Exemples de EON</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f9;

color: #333;

margin: 0;

padding: 0;

}

header {

background-color: #4CAF50;

color: white;

text-align: center;

padding: 20px;

}

.content {

padding: 20px;

}

.code {

background-color: #282c34;

color: white;

padding: 10px;

border-radius: 5px;

margin: 10px 0;

font-family: 'Courier New', Courier, monospace;

}

.output {

background-color: #f1f1f1;

padding: 10px;

margin: 10px 0;

border-radius: 5px;

}

</style>

</head>

<body>

<header>

<h1>Exemples de Code EON</h1>

</header>

<div class="content">

<h2>Exemple 1 : Somme des carrés</h2>

<div class="code">

fn somme\_des\_carres(n: int) -> int {<br>

&nbsp;&nbsp;let somme = 0<br>

&nbsp;&nbsp;for i in range(1, n+1) {<br>

&nbsp;&nbsp;&nbsp;&nbsp;somme += i \* i<br>

&nbsp;&nbsp;}<br>

&nbsp;&nbsp;return somme<br>

}<br><br>

let resultat = somme\_des\_carres(10)<br>

print(resultat)

</div>

<div class="output">

Résultat : 385

</div>

<h2>Exemple 2 : Calcul parallèle des carrés</h2>

<div class="code">

let résultats = [1, 2, 3, 4, 5].parallel\_map(fn(x) -> x \* x)<br>

print(résultats)

</div>

<div class="output">

Résultat : [1, 4, 9, 16, 25]

</div>

<h2>Exemple 3 : Nombres premiers jusqu'à 50</h2>

<div class="code">

fn est\_premier(n: int) -> bool {<br>

&nbsp;&nbsp;if n <= 1 {<br>

&nbsp;&nbsp;&nbsp;&nbsp;return false<br>

&nbsp;&nbsp;}<br>

&nbsp;&nbsp;for i in range(2, n) {<br>

&nbsp;&nbsp;&nbsp;&nbsp;if n % i == 0 {<br>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;return false<br>

&nbsp;&nbsp;&nbsp;&nbsp;}<br>

&nbsp;&nbsp;}<br>

&nbsp;&nbsp;return true<br>

}<br><br>

let primes = []<br>

for i in range(2, 50) {<br>

&nbsp;&nbsp;if est\_premier(i) {<br>

&nbsp;&nbsp;&nbsp;&nbsp;primes.append(i)<br>

&nbsp;&nbsp;}<br>

}<br>

print(primes)

</div>

<div class="output">

Résultat : [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

</div>

</div>

</body>

</html>