PROJET D'EXAMEN · MULTI-LANGAGES

Projet 10
Calculatrice scientifique distribuée

Évaluation d'expressions avec historique persistant

Domaine :	Mathématiques · Outils utilitaires
Composantes :	C (DLL) · Python (module + tests) · Flask (API + dashboard) · C# (GUI)
Cours :	5e TTR Informatique · 6e TTR Informatique
Format :	Projet individuel
Édition :	19 mai 2026

1 Contexte et objectifs

Tu réalises une calculatrice scientifique qui sait évaluer des expressions mathématiques simples : opérations de base, racine carrée, puissance, factorielle, PGCD, PPCM, logarithmes, fonctions trigonométriques. L'API Flask garde l'historique de tous les calculs. Le dashboard montre les fonctions les plus utilisées et l'activité par heure.

Objectifs pédagogiques

- Compiler une bibliothèque C en DLL et l'appeler depuis Python via `ctypes`.
- Écrire un module Python testable, avec validation et persistance des données.
- Concevoir une API REST Flask et un dashboard Chart.js qui la consomme.
- Développer une interface C# Windows Forms qui communique avec l'API.
- Comparer trois approches de calcul (Python pur, ctypes scalaire, ctypes batch).

2 Cahier des charges fonctionnel

L'application doit permettre à l'utilisateur de :

- Opérations de base : +, −, ×, ÷.
- Opérations scientifiques : sqrt, power, factorial, gcd, lcm, log10, sin, cos, tan.
- Parser d'expression rudimentaire (un opérateur ou une fonction par expression).
- Historique persistant des 1000 derniers calculs.
- Export historique au format CSV ou JSON.
- Statistiques : fonctions les plus utilisées, calculs par heure.
- Gestion des erreurs : division par zéro, racine de négatif, factorielle > 20…

> **Format d'expression accepté —** function(arg) ex : sqrt(144), factorial(5) function(arg1, arg2) ex : power(2, 10), gcd(48, 36) arg1 OP arg2 ex : 12 + 5, 3.14 * 2

> **Gestion d'erreurs —** division par zéro · factorielle > 20 (overflow long long) racine d'un négatif · log d'un nombre ≤ 0 expression non parsable → 400 avec message clair

3 Module C — `calculs.c` → `calculs.dll`

Les fonctions ci-dessous sont à exporter dans la DLL. Chaque fonction doit être présente en version **scalaire** (pour l'application normale) ; les fonctions marquées *batch* servent au benchmark de la Partie 5.

Signature	Rôle
`int calc_gcd(int a, int b);`	PGCD par Euclide
`int calc_lcm(int a, int b);`	PPCM = a × b / pgcd(a, b)
`long long calc_factorial(int n);`	n ≤ 20 (sinon overflow)
`double calc_power(double base, int exp);`	base^exp par exponentiation rapide
`double calc_sqrt(double x);`	Wraps sqrt() avec garde x ≥ 0
`double calc_log10(double x);`	Wraps log10() avec garde x > 0
`double calc_sin(double radians);`	Trigonométrie
`double calc_cos(double radians);`	Trigonométrie
`void calc_power_batch(double* b, int* e, double* out, int n);`	Puissance batch

Compilation Windows : `gcc -shared -o calculs.dll calculs.c -lm`

4 Module Python — `module.py`

Le module Python est le seul pont entre la DLL et le reste de l'application. Il contient les wrappers `ctypes` et la logique métier (validation, persistance, statistiques).

- `gcd(a, b), lcm(a, b), factorial(n) — wrappers ctypes`
- `power(base, exp), sqrt(x), log10(x) — wrappers ctypes`
- `parse_expression(expr: str) -> tuple — (op, args)`
- `evaluate(expr: str) -> float — orchestrateur`
- `save_calculation(expr, result, op) -> dict`
- `get_history(limit=100, offset=0) -> list`
- `clear_history()`
- `get_top_functions(n=5) -> list`
- `get_activity_by_hour() -> dict`

5 API Flask — `app.py`

Toutes les réponses respectent le format `{"status": "ok|error", "data": ...}`. Statuts HTTP : 200 (lecture), 201 (création), 400 (validation), 404 (non trouvé), 500 (erreur serveur).

Méthode	Chemin	Description
POST	`/api/calculate`	Corps : {expression}. Retourne result + opération détectée
GET	`/api/history?limit=&offset=`	Historique paginé
DELETE	`/api/history`	Vide l'historique
GET	`/api/stats/top-functions?n=5`	Top fonctions
GET	`/api/stats/activity-by-hour`	Histogramme par heure

6 Dashboard web — `/dashboard`

Dashboard lecture seule, rafraîchissement automatique toutes les 30 secondes via fetch JS. Trois graphiques Chart.js obligatoires :

1. Fonctions les plus utilisées (bar chart horizontal).
2. Activité par heure de la journée (line chart 24 points).
3. Répartition par type d'opération (doughnut : arithmétique / scientifique / trig).

7 Interface C# — Windows Forms

Application Windows Forms qui communique avec l'API uniquement via `HttpClient`. Aucun calcul ne doit être fait côté C#.

- MainForm — clavier scientifique (Buttons 0-9, +, −, ×, ÷, sqrt, pow, !...).
- HistoryForm — liste paginée + bouton « Vider ».
- ExportForm — boutons Export CSV / JSON.
- SettingsForm — précision d'affichage (nombre de décimales).

8 Format des données

Variante 5e — Persistance JSON

```
{
  "calculation": {
    "id": 42,
    "expression": "sqrt(144)",
    "operation": "sqrt",
    "result": 12.0,
    "timestamp": "2026-05-19T10:30:00"
  }
}
```

Variante 6e — Persistance SQL (MySQL/SQLite)

```
CREATE TABLE calculations (
  id         INTEGER PRIMARY KEY AUTOINCREMENT,
  expression TEXT NOT NULL,
  operation  TEXT NOT NULL,
  result     REAL,
  error      TEXT,
  timestamp  TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_calc_timestamp ON calculations(timestamp);
CREATE INDEX idx_calc_operation ON calculations(operation);
```

9 Plan de tests Pytest

Minimum 10 tests Pytest dans `tests_module.py`. Tous doivent retourner **PASSED**.

1. `test_gcd_basic — gcd(48, 36) == 12`
2. `test_gcd_coprime — gcd(7, 13) == 1`
3. `test_lcm_basic — lcm(4, 6) == 12`
4. `test_factorial_5 — 120`
5. `test_factorial_20 — 2432902008176640000`
6. `test_power_basic — power(2, 10) == 1024`
7. `test_sqrt_perfect — sqrt(144) == 12.0`
8. `test_parse_expression_function_one_arg`
9. `test_parse_expression_function_two_args`
10. `test_evaluate_division_by_zero_raises`

10 Critères de validation et remise

Barème (sur 24)

Composante	Points
Module C — fonctions scalaires + batch, compile sans warning	4
Module Python — wrapper ctypes + logique métier	4
Tests Pytest — 10 tests PASSED	3
API Flask — tous les endpoints fonctionnels	4
Dashboard web — 3 graphiques opérationnels	3
GUI C# — communication API et écrans demandés	4
Benchmark — 3 scénarios + tableau + explication orale	2
Total	**/24**

Structure du dossier de remise

```
projet-10-calculatrice/
├── calculs.c
├── calculs.dll
├── module.py
├── tests_module.py
├── app.py
├── generator.py
├── benchmark.py
├── templates/
│   └── dashboard.html
├── data/                        (5e – JSON)   ou   db.sqlite   (6e – SQL)
└── GUI/
    └── Projet10.sln
```

Démonstration orale (5 minutes)

1. Lance ton API Flask et ton dashboard.
2. Lance le générateur de données et observe le dashboard se remplir.
3. Ouvre la GUI C# et démontre 2 cas d'usage.
4. Présente le résultat du benchmark et explique pourquoi le scénario batch est plus rapide.