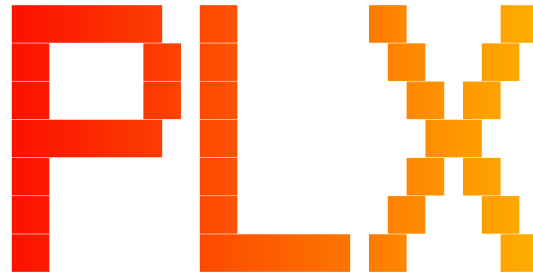


# Concevoir une expérience d'apprentissage interactive à la programmation avec PLX



**Travail de Bachelor - Samuel Roland - 2025**

**Suivi par Bertil Chapuis**

- Programmation complexe et abstraite
- Pratique et feedback
- Code fonctionnel vs code de qualité
- Lenteur des vérifications manuelles

```
chien-du-quartier> ls
main.c
chien-du-quartier> gcc main main.c
/usr/bin/ld: cannot find main: No such file or
directory
collect2: error: ld returned 1 exit status
chien-du-quartier> gcc -o main main.c
chien-du-quartier> main
bash: main: command not found...
chien-du-quartier> ./main
Trop d info
chien-du-quartier> ./main Dogy
Trop d info
Le chien Dogy est sympa
chien-du-quartier> ????
```

**Et PLX dans ce contexte ?**

## Comment faciliter la rédaction et la maintenance des exercices ?

- PRG1: 100+ exercices de code
- Beaucoup de temps de rédaction
- Formats textuels appréciés (PRG1, PRG2, WEB, DAI, AMT)
- Git, collaboration, IDE local
- Décrire le cours, les compétences et les exercices

**Quel format textuel choisir pour optimiser la rédaction ?**

## Salue-moi

Un petit programme qui te salue avec ton nom complet.

Assure-toi d'avoir la même sortie que ce scénario, en répondant `John` et `Doe` manuellement.

```
> ./main
Quel est ton prénom ? John
Salut John, quel est ton nom de famille ? Doe
Passe une belle journée John Doe !
>
```

Démarre avec ce bout de code.

```
int main(int argc, char *argv[]) {
    // ???
}
```

Vérifie que ton programme ait terminé avec le code de fin 0, en lançant cette commande.

```
> echo $?
0
```

## # Salue-moi

Un petit programme qui te salue avec ton nom complet.

Assure-toi d'avoir la même sortie que ce scénario, en répondant `John` et `Doe` manuellement.

```
```
```

```
> ./main
```

```
Quel est ton prénom ? John
```

```
Salut John, quel est ton nom de famille ? Doe
```

```
Passe une belle journée John Doe !
```

```
>
```

```
```
```

Démarre avec ce bout de code.

```
```c
```

```
int main(int argc, char *argv[]) {
```

```
    // ???
```

```
}
```

```
```
```

Vérifie que ton programme ait terminé avec le code de fin 0, en lançant cette commande.

```
```sh
```

```
> echo $?
```

```
0
```

```
```
```

<details>

<summary>Solution</summary>

```
```c
```

```
#include <stdio.h>
```

```
// suite du code
```

```
```
```

</details>

```
{
  "exo": "Salue-moi",
  "instruction": "Un petit programme qui te salue avec ton nom complet.",
  "checks": [
    {
      "name": "Il est possible d'être salué avec son nom complet",
      "sequence": [
        { "kind": "see", "value": "Quel est ton prénom ?" },
        { "kind": "type", "value": "John" },
        { "kind": "see", "value": "Salut John, quel est ton nom de famille ?" },
        { "kind": "type", "value": "Doe" },
        { "kind": "see", "value": "Passe une belle journée John Doe !" },
        { "kind": "exit", "code": 0 }
      ]
    }
  ]
}
```

Equivalent JSON

```
exo: Salue-moi
instruction: Un petit programme qui te salue avec ton nom complet.
checks:
  - name: Il est possible d'être salué avec son nom complet
    sequence:
      - kind: see
        value: Quel est ton prénom ?
      - kind: type
        value: John
      - kind: see
        value: Salut John, quel est ton nom de famille ?
      - kind: type
        value: Doe
      - kind: see
        value: Passe une belle journée John Doe !
      - type: exit
        value: 0
```

Equivalent YAML



```
package {
  name my-pkg
  version "1.2.3"

  dependencies {
    // Nodes can have standalone values as well as
    // key/value pairs.
    lodash "^3.2.1" optional=#true alias=underscore
  }

  scripts {
    // "Raw" and dedented multi-line strings are supported.
    message """
      hello
      world
      """
    build #"""
      echo "foo"
      node -c "console.log('hello, world!');"
      echo "foo" > some-file.txt
      """"#
    }
  }
}
```

Exemple de la syntaxe KDL pour définir un `package.json` (1)

```
[.multiple-choice-1]
[!What color is milk?]
[?Cows produce milk.]
[+white]
[-red]
[-blue]
```

Choix multiple défini en syntaxe Bitmark (2).

## Comment faciliter la rédaction et la maintenance des exercices ?

En créant une nouvelle syntaxe textuelle

- concise
- facile à rédiger
- tolérante aux erreurs
- avec validation intégrée
- et validation avancée en Rust

Dans un fichier `exo.dy`

**exo** Salue-moi

Un petit programme qui te salue avec ton nom complet.

**check** Il est possible d'être salué avec son nom complet

**see** Quel est ton prénom ?

**type** John

**see** Salut John, quel est ton nom de famille ?

**type** Doe

**see** Passe une belle journée John Doe !

**exit** 0

Dans un fichier `course.dy`

```
// Définition du cours
```

```
course Programmation 1
```

```
code PRG1
```

```
goal Apprendre des bases solides du C++
```

Dans un fichier `skills.dy`

**skill** Introduction

**dir** intro

**skill** Enumerations

**dir** enums

**skill** Structures

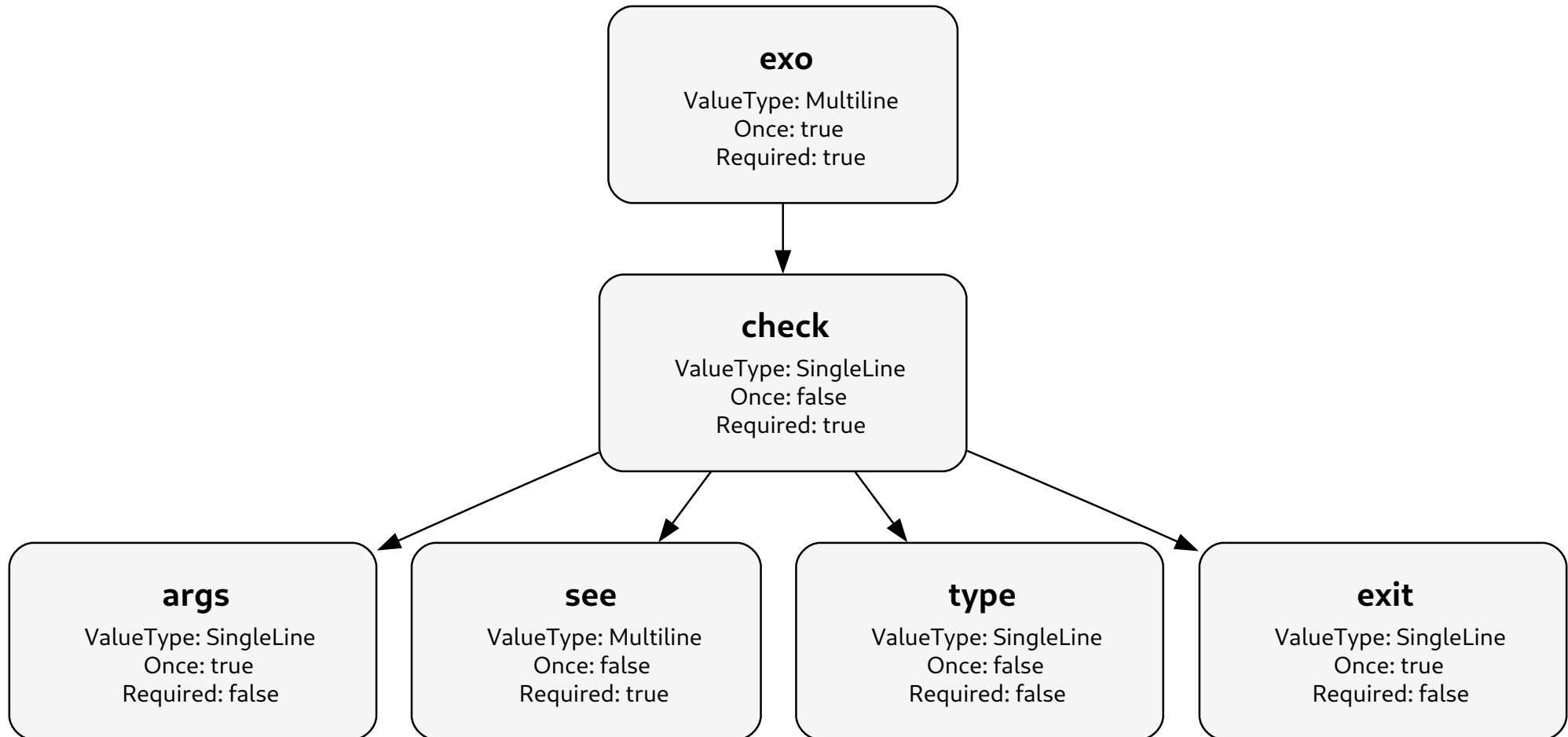
**dir** structs

**skill** Pointers and memory

**dir** pointers

**skill** Parsing

**dir** parsing



- Parseur Rust - librairie `dy`
- Définitions des clés PLX - librairie `plx-dy`
- Intégration de `plx-dy` à PLX desktop
- Intégration de `plx-dy` au CLI `plx parse`

# Démo de la syntaxe DY



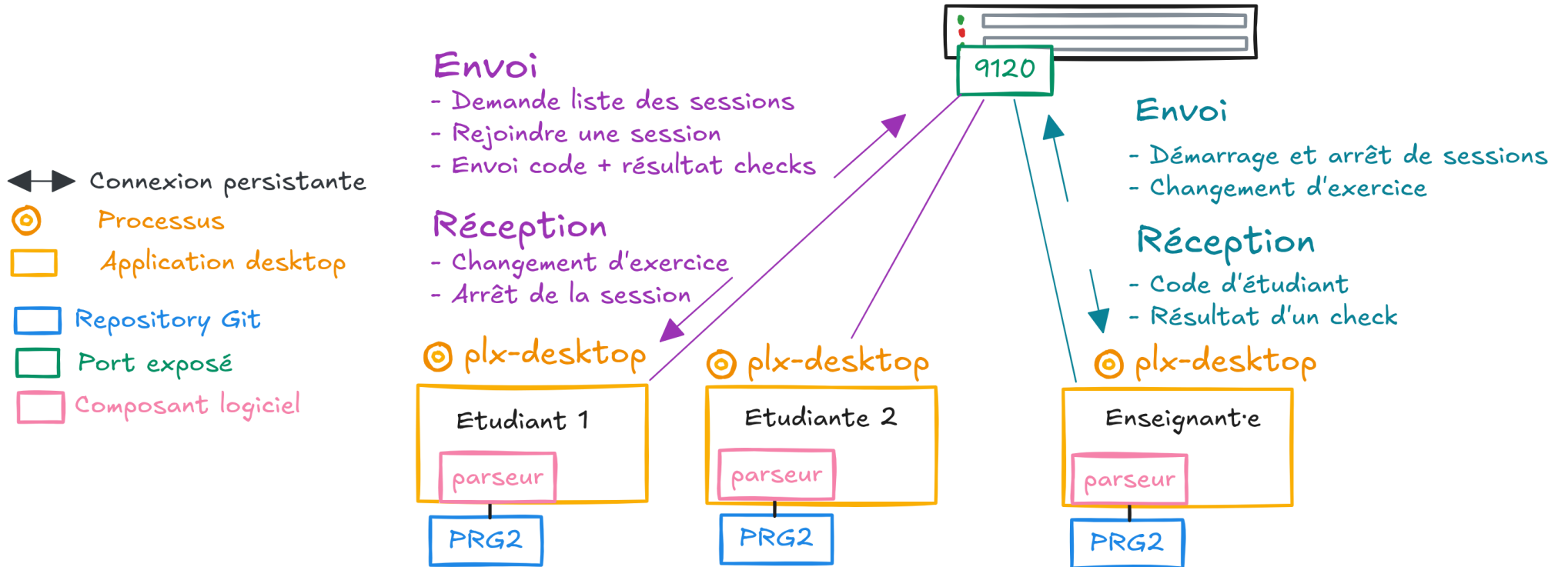
## **Problèmes lors des séances d'entraînement en classe**

- Est-ce que c'est acquis ?
- Est-ce que ma classe progresse ?
- Peu de questions
- Confusion ou blocage
- Le code fonctionne
- Feedback de qualité de code

**Comment les enseignant·es peuvent voir le code et les résultats en temps réel ?**

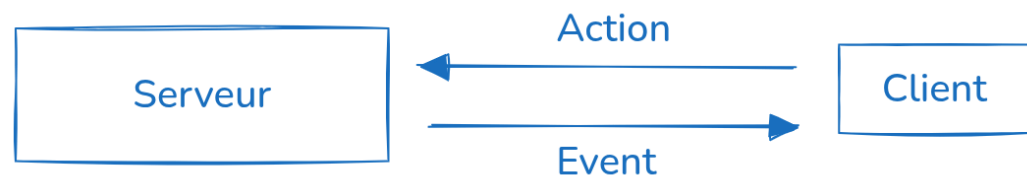
Solution: Les sessions live hébergées sur un serveur central

Serveur PLX live.plx.rs ☉ plx server



- Formats textuels et binaires: XML, JSON, ProtoBuf, MessagePack, ...
- Protocoles de transport bidirectionnel: WebSockets, gRPC, tarpc, ...
- JSON et WebSockets
- Pas de compte: `client_id` et `client_num`
- Système de rôle: `Leader` et `Follower`

```
{  
  "type": "ExoSwitched",  
  "content": {  
    "path": "intro/salue-moi"  
  }  
}
```



```
{
  "type": "SendFile",
  "content": {
    "path": "main.c",
    "content": "\n#include
<stdio.h>\n\nint main(int argc, char
*argv[]) {\n    printf(\"hello
world!\n\n\");\n    return 0;\n}\n"
  }
}
```

```
{
  "type": "ForwardFile",
  "content": {
    "client_num": 23,
    "file": {
      "path": "main.c",
      "content": "\n#include
<stdio.h>\n\nint main(int argc, char
*argv[]) {\n    printf(\"hello
world!\n\n\");\n    return 0;
\n}\n\n",
      "time": 1751632509
    }
  }
}
```

```
{
  "type": "Stats",
  "content": {
    "followers_count": 32,
    "leaders_count": 2
  }
}
```

```
{
  "type": "Error",
  "content": {
    "type": "FailedToJoinSession",
    "reason": "No session found with this name in this group id"
  }
}
```

- Serveur en Rust intégré à `plx server`
- Runtime Tokio et `tokio-tungstenite`
- Gestion des sessions
- Dashboard pour les enseignant·es

PRG1  
Programmation 1


PRG2  
Programmation 2

DEMO  
PLX demo course

Join one of the live session

1. Alice




2. Jack






# Démo d'une session live











## Objectifs principaux

-  Un serveur en Rust lancé via le CLI plx permettant de gérer des sessions live.
-  Une librairie en Rust de parsing de la syntaxe DY.
-  Une intégration de cette librairie dans PLX.



## Objectifs fonctionnels

-  Tous les objectifs atteints
-  Sauf la vue globale des checks sur tous les exercices
-  Intégration desktop des sessions

## Objectifs non-fonctionnels

-  Supporter les déconnexions temporaires
-  Le serveur doit supporter 300 connexions persistantes simultanées
-  Une session live s'arrête automatiquement après 30 minutes
-  Aucun code externe ne doit être exécuté automatiquement par PLX
-  < à 3s pour l'envoi et la réception d'un check
-  Tests de bout en bout avec multiples clients
-  Vitesse du parseur DY - 200 exercices en < 1s
-  Retranscrire un exo en < 1min

## Objectifs "nice-to-have"

-  Intégration des erreurs dans un serveur de langage dans VSCode et Neovim.
-  Autogénérer des définitions TreeSitter depuis les clés

- Supporter l'exécution de `type` et `exit`
- Améliorer l'envoi des résultats des checks
- Gérer les pannes des clients
- Étendre la syntaxe DY
- Améliorer le dashboard

# Programmation 1

## Skills

|    |                                |
|----|--------------------------------|
| 1  | Introduction                   |
| 2  | Elements de base               |
| 3  | Structures de Controle         |
| 4  | Fonctions                      |
| 5  | Flux                           |
| 6  | Types arithmétiques et...      |
| 7  | Structure_Enum                 |
| 8  | Chaines de caracteres          |
| 9  | Tableaux                       |
| 10 | Surcharge et Genericite        |
| 11 | Classes                        |
| 12 | iterator - algorithm - numeric |
| 13 | Classe générique               |
| 14 | Gestion des erreurs            |
| 15 | Allocation dynamique           |
| 16 | Exercices récapitulatifs       |

## 61 Exos

|    |                              |            |
|----|------------------------------|------------|
| 0  | 9.1 Calcul vectoriel         | Todo       |
| 0  | 9.2 Stack                    | Todo       |
| 0  | 9.3 Jour de la semaine       | Todo       |
| 0  | 9.4 resize, push_back,...    | Todo       |
| 0  | 9.5 Suppression du ou des... | Todo       |
| 0  | 9.6 Suppression d'une valeur | Todo       |
| 1  | 9.7 Rendre unique            | Todo       |
| 1  | 9.8 Moyennes de notes        | InProgress |
| 16 | 9.9 Remplacement de vale...  | Todo       |
| 4  | 9.10 Eléments strictement... | Todo       |
| 14 | 9.11 Ensemble                | Todo       |
| 5  | 9.12 Ensemble trié           | Todo       |
| 3  | 9.13 Recherche...            | InProgress |
| 2  | 9.14 Matrice et vecteur      | Todo       |
| 3  | 9.15 Matrice de caractères   | Todo       |
| 12 | 9.16 Médailles...            | InProgress |

## Jour de la semaine

Soient les deux déclarations

```
enum class Day { ERROR, LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,
SAMEDI, DIMANCHE};
const std::array<string, 8> DAY{"ERROR", "LUNDI", "MARDI", "MERCREDI", "JEUDI",
"VENDREDI", "SAMEDI", "DIMANCHE"};
```

Ecrire les deux fonctions

- **stringToDay** : reçoit un jour en *string* et retourne l'équivalent en *enum*
- **dayToString** : reçoit un jour en *enum* et retourne l'équivalent en *string*

Le cas échéant, retourne *enum ERROR* ou *string "ERROR"* selon la fonction.

**NB** ne pas utiliser de *switch* ni le *find* de la librairie *algorithm*.

## Checks

### C1: Conversion et gestion des erreurs fonctionels

Expected

```
day_to_string(Day(0)) => "ERROR"
day_to_string(Day(1)) => "LUNDI"
day_to_string(Day(2)) => "MARDI"
day_to_string(Day(3)) => "MERCREDI"
day_to_string(Day(4)) => "JEUDI"
day_to_string(Day(5)) => "VENDREDI"
day_to_string(Day(6)) => "SAMEDI"
day_to_string(Day(7)) => "DIMANCHE"
day_to_string(Day(9)) => "ERROR"
```

1. MARCHÁN, Kat. KDL, a cuddly document language. En ligne. 2025. Disponible à l'adresse: <https://kdl.dev/>
2. BITMARK ASSOCIATION. Quizzes - .multiple-choice, .multiple-choice-1. En ligne. 2025. Disponible à l'adresse: <https://docs.bitmark.cloud/quizzes/#multiple-choice-multiple-choice-1>

## Configuration

```
// Number of files saved per minutes per client
const MIN_SAVE_PER_MINUTE: u16 = 2;
const MAX_SAVE_PER_MINUTE: u16 = 20;
// Number of client to put in a live session
const MIN_CLIENT_PER_SESSION: u16 = 20;
const MAX_CLIENT_PER_SESSION: u16 = 60;
```

## Situation de départ

700Ko of RAM + 0% CPU

```
> docker stats env-plx-1
```

| CONTAINER ID | NAME      | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O        | BLOCK I/O | PIDS |
|--------------|-----------|-------|-------------------|-------|----------------|-----------|------|
| e692576c9958 | env-plx-1 | 0.00% | 708KiB / 1.91GiB  | 0.04% | 5.2kB / 1.56kB | 0B / 0B   | 2    |

Avec 30 connexions TCP existantes

```
> ss -s
```

```
Total: 217
```

```
TCP: 30 (estab 3, closed 14, orphaned 1, timewait 0)
```

## En ajoutant les 400 clients

53MB de RAM et 0.68% de CPU.

```
> docker stats env-plx-1
```

| CONTAINER ID | NAME      | CPU % | MEM USAGE / LIMIT  | MEM % | NET I/O       | BLOCK I/O | PIDS |
|--------------|-----------|-------|--------------------|-------|---------------|-----------|------|
| e692576c9958 | env-plx-1 | 0.68% | 52.84MiB / 1.91GiB | 2.70% | 882kB / 247kB | 0B / 0B   | 2    |

Le 30 est passé à 423.

```
> ss -s
```

```
Total: 217
```

```
TCP: 423 (estab 2, closed 407, orphaned 7, timewait 0)
```

La latence mesurée visuellement d'un changement d'exercice reste inférieur à une seconde.

```
use plx_dy::parse_course;

fn main() {
    let filename = &Some("course.dy".to_string());
    let course_text = "course Programmation 2
code PRG2
goal";
    let result = parse_course(filename, course_text);
    dbg!(result);
}
```

```
ParseResult {
  items: [
    DYCourse {
      name: "Programmation 2",
      code: "PRG2",
      goal: "",
    },
  ],
  errors: [
    ParseError {
      range: Range {
        start: Position {
          line: 2, character: 4 },
        end: Position {
          line: 2, character: 4 },
      },
      error: MissingRequiredValue("goal"),
    },
  ],
  some_file_path: Some("course.dy"),
  some_file_content: Some("course Programmation
2\ncode PRG2\ngoal"),
}
```