EasyTracker

Une bibliothèque pour contrôler, inspecter et visualiser l'exécution d'un programme

Théo Barollet, Florent Bouchez Tichadou, François Broquedis, Fabrice Rastello, Manuel Selva
Équipe CORSE

12 mai 2022

Contexte : enseignement de la programmation

Visualisation de l'exécution d'un programme

- "How does a novice programmer (or an increasingly expert programmer) know where to look?" [Fincher et al. 2020]
- Depuis les années 70 [Sorva, Karavirta, and Malmi 2013]

Contexte : enseignement de la programmation

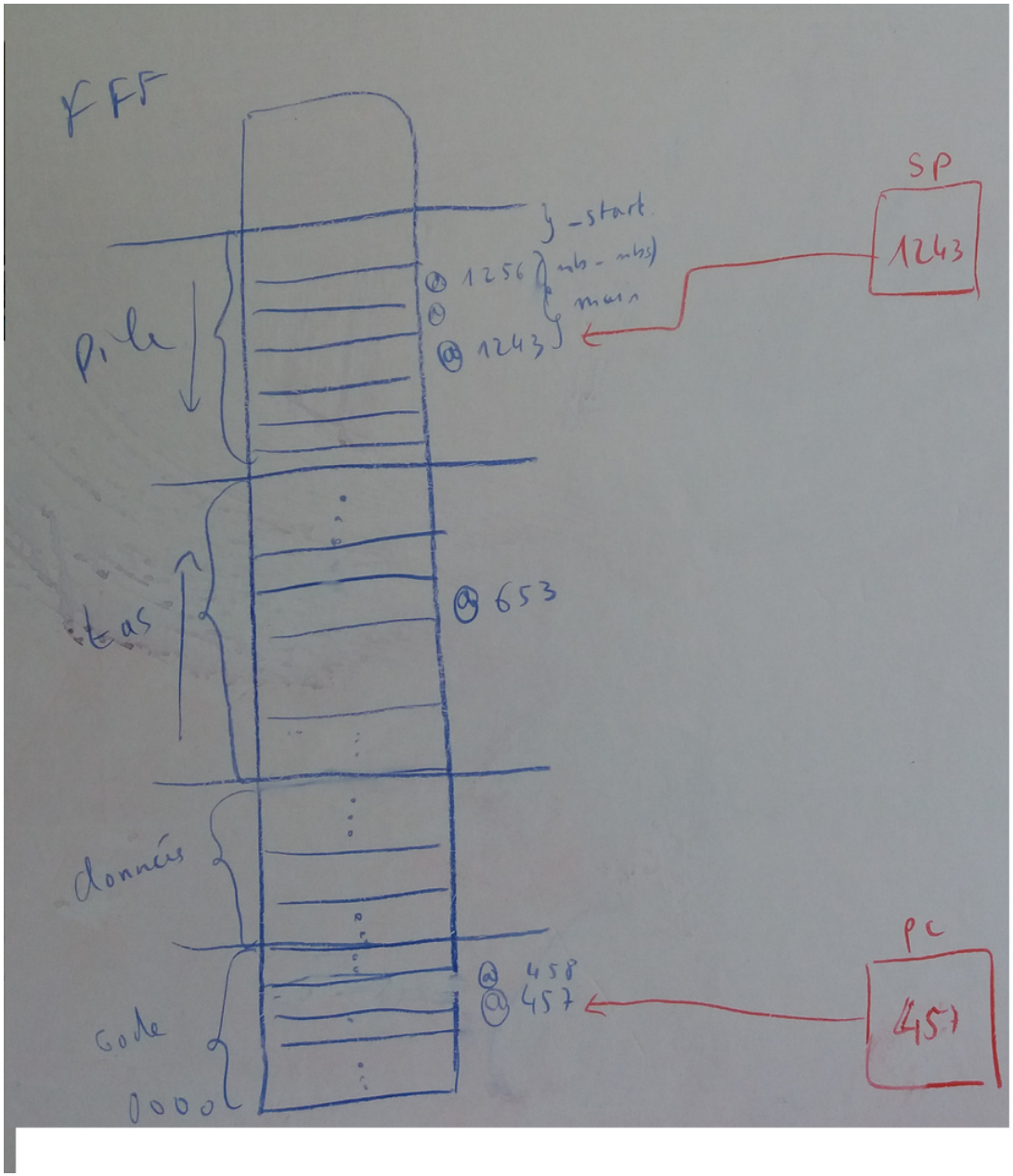
Visualisation de l'exécution d'un programme

- "How does a novice programmer (or an increasingly expert programmer) know where to look?" [Fincher et al. 2020]
- Depuis les années 70 [Sorva, Karavirta, and Malmi 2013]

Machine notionnelle [Boulay, O'Shea, and Monk 1981; Robins, Rountree, and Rountree 2003; Sorva 2013]

- Définition : "a pedagogic device to assist the understanding of some aspect of programs or programming" [Fincher et al. 2020]
- Très souvent associée à une représentation visuelle [Notional machines: A Curated Collection 2020]

Motivations : les dessins à la main au tableau sont nécessaires mais ...



Avantages

- Étudiant : déroulé "lent"
- Enseignant : gratuit

Inconvénients

- Étudiant : non reproductible
- Enseignant : pas distribuable
- Enseignant : non éditable
- Enseignant : pénible à dessiner

Motivations : les dessins à la main sur ordinateur sont pas mal mais ...

TD9. Classes, instances et références

BPI : c'est en programmant qu'on devient programmeur (mais pas que)

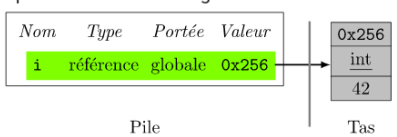
- 1- Bases
- 2- Itérations
- 3- Références
  - Sommaire
  - CM3
  - TD
    - TD9. Classes, instances et références
    - TD10. Listes simplement chaînées
    - TD11. Exceptions, files et piles
    - TD12. Redonnons la main
    - TD13. Listes doublement chaînées
  - TP
- 4- Récursivité
- Projet
- Examen mi-parcours
- Examen final

```python
#!/usr/bin/env python3

"""Un petit programme pour nous, un grand
   programme pour l'interpréteur"""
i = 42
j = i
k = 41
k += 1
```

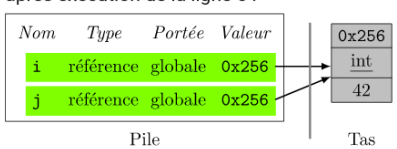
Cliquez ici pour révéler la correction.

Voici les quatre schémas corrects représentant l'état du programme :

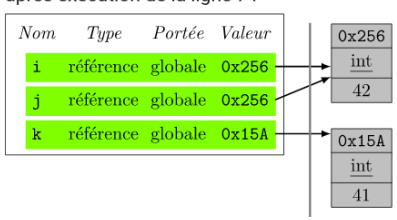
après exécution de la ligne 5 :



après exécution de la ligne 6 :



après exécution de la ligne 7 :



Avantages

- Étudiant : schémas propres
- Enseignant : distribuable
- Enseignant : éditable

Inconvénients

- Étudiant : non reproductible
- Enseignant : **ultra** pénible à dessiner

Motivations : les dessins à la main sur ordinateur sont pas mal mais ...

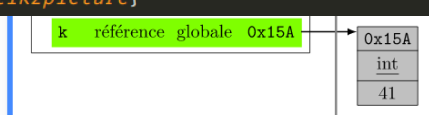
TD9. Classes, instances et références

BPI : c'est en programmant qu'on devient programmeur (mais pas que)

- 1- Bases
- 2- Itérations
- 3- Références
  - Sommaire
  - CM3
  - TD
    - TD9. Classes, instances et références
    - TD10. Listes simplement chaînées
    - TD11. Exceptions, files et piles
    - TD12. Redonnons la main
    - TD13. Listes doublement chaînées
  - TP
- 4- Récursivité
- Projet
- Examen mi-parcours
- Examen final

```python
#!/usr/bin/env python3

"""Un petit programme pour nous, un grand
   programme pour l'interpréteur"""
```

```
\begin{tikzpicture}[font=\normalsize]

  % Dessin du tableau des variables
  \matrix[tvars] (tvariables) {
      \textit{Nom}&\textit{Type}&\textit{Portée}&\textit{Valeur}\\
      \texttt{i}&référence&globale&\texttt{0×256}\\
      \texttt{j}&référence&globale&\texttt{0×256}\\
      \texttt{k}&référence&globale&\texttt{0×15A}\\
  };
  \colorfillrows{tvariables}{4}{4}{chartreuse}% 1:matrixname, 2:nb_rows, 3:color

  % Dessin des instance
  \coordinate (ijpos) at ($(tvariables.north east) + (15mm, -2mm)$);
  \drawint{ijpos}{north}{ij}{0×256}{42}% 1:pos, 2:anchor, 3:name, 4:addr, 5:val
  \coordinate (kpos) at ($(ij.south) - (0mm, 5mm)$);
  \drawint{kpos}{north}{k}{0×3B5}{17}

  % Lien entre tableau des variables et instances
  \draw[ref] (tvariables-2-4) -- (ij.west |- tvariables-2-4);
  \draw[ref] (tvariables-3-4.east) -- ($(ij.west |- tvariables-2-4) -(0mm, 2mm)$);
  \draw[ref] (tvariables-4-4) -- (k.west |- tvariables-4-4);

  % Stack vs heap
  \node[fill=none, draw=none, right=of tvariables.north east, anchor=north west] (e) {};
  \path (tvariables.north east) -- node[](mid){} (e);
  \coordinate (sepsouth) at ($(mid |- ij.south) - (0mm, 28mm)$);
  \draw[gray, very thick] (mid) -- (sepsouth);
  \node at (tvariables |- sepsouth) {Pile};
  \node at (ij |- sepsouth) {Tas};
\end{tikzpicture}
```



as propres

ribuable

able

productible

a pénible à dessiner

Motivations : les dessins semi-générés [Wagner 2017] c'est trop cool mais...

Listes

Afin de vous aider à déboguer votre code, on vous fournit le module tycat.py. Il permet de tracer tous les appels à une fonction/méthode graphiquement dans *terminology* (ne pas utiliser un autre émulateur de terminal).

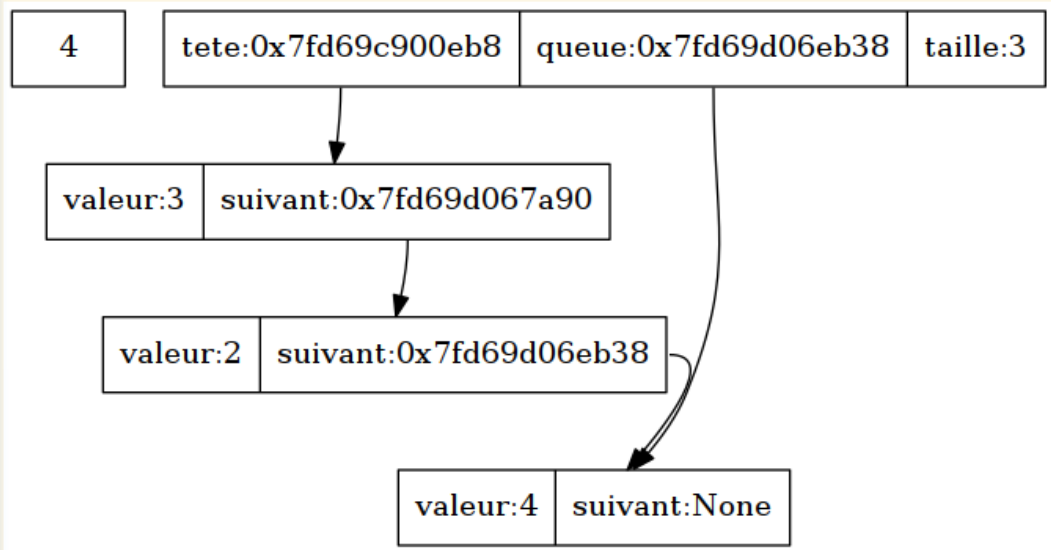
Pour s'en servir, il suffit de :

- ajouter **from** `tycat` **import** `trace` au début de votre code ;
- ajouter une ligne `@trace` avant chaque déclaration de fonction que vous voulez tracer.

Le module affiche tous les arguments et toutes les valeurs renvoyées lors de chaque appel à chaque fonction tracée. Vous pouvez donc ajouter un **return** `self` à la fin d'une méthode pour visualiser l'état de l'objet avant et après chaque appel.

Voici un petit exemple d'une trace lors d'un appel supprimant '4' d'une liste.

```
supprimer((3,2,4),4)
```



Avantages

- Étudiant : schémas propres
- Étudiant : reproductible
- Enseignant : distribuable
- Enseignant : éditable

Inconvénients

- Étudiant : instrumentation nécessaire
- Enseignant : instrumentation nécessaire

Contribution : une bibliothèque pour les enseignants

API indépendante du langage du programme : Python / C

- Contrôle : `start`, `break`, `watch`, `continue`, `next`, ...
- Inspection : `get_current_frame`, `get_globals`, ...
- Visualisation : `show_stack`, `show_heap`, `show_source`, ...

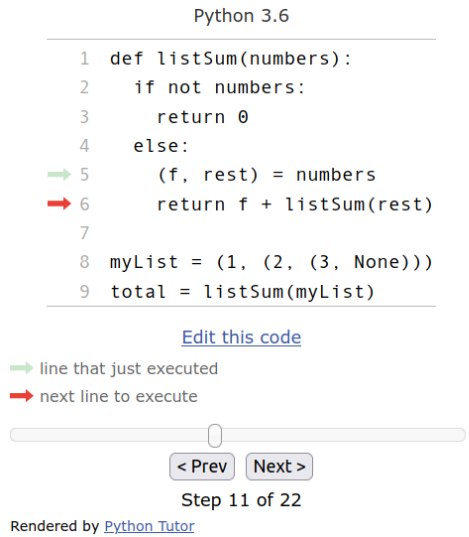
Contribution : une bibliothèque pour les enseignants

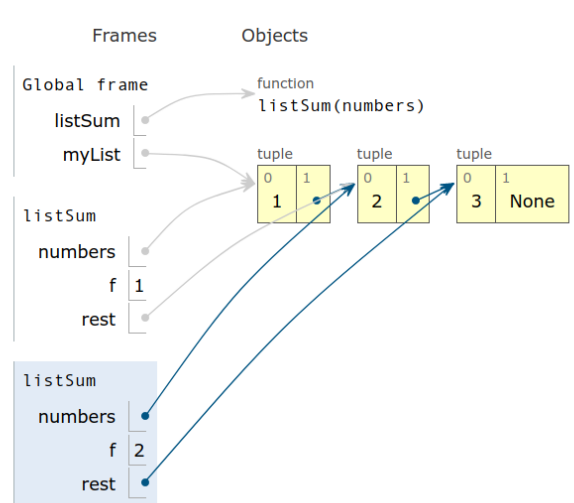
API indépendante du langage du programme : Python / C

- Contrôle : `start`, `break`, `watch`, `continue`, `next`, ...
- Inspection : `get_current_frame`, `get_globals`, ...
- Visualisation : `show_stack`, `show_heap`, `show_source`, ...

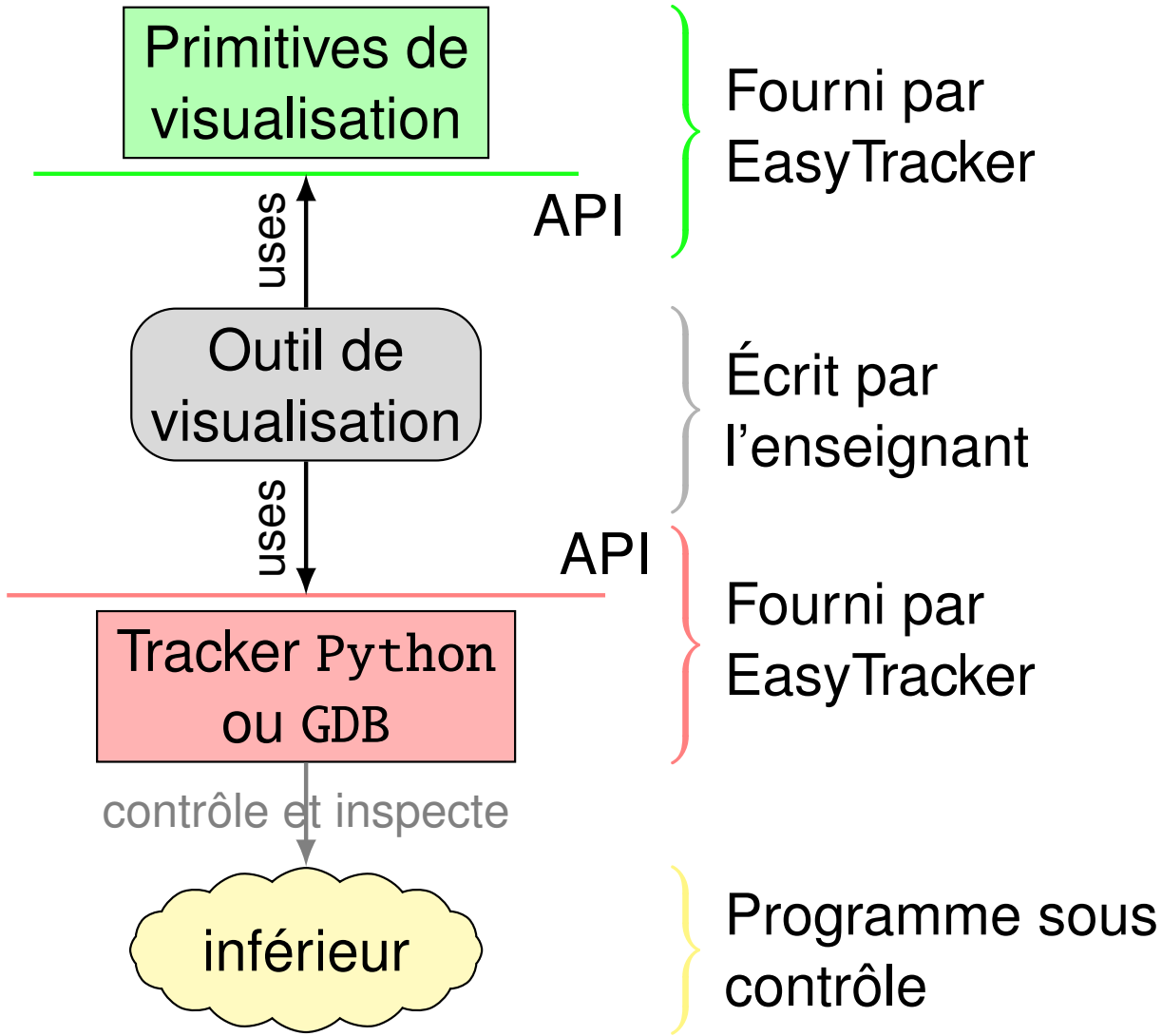
Mais ça n'existe pas déjà ça ?

- *Python Tutor* [Guo 2013]
- Différences
  - Pas personnalisable
  - Trace complète





Contribution : architecture logicielle

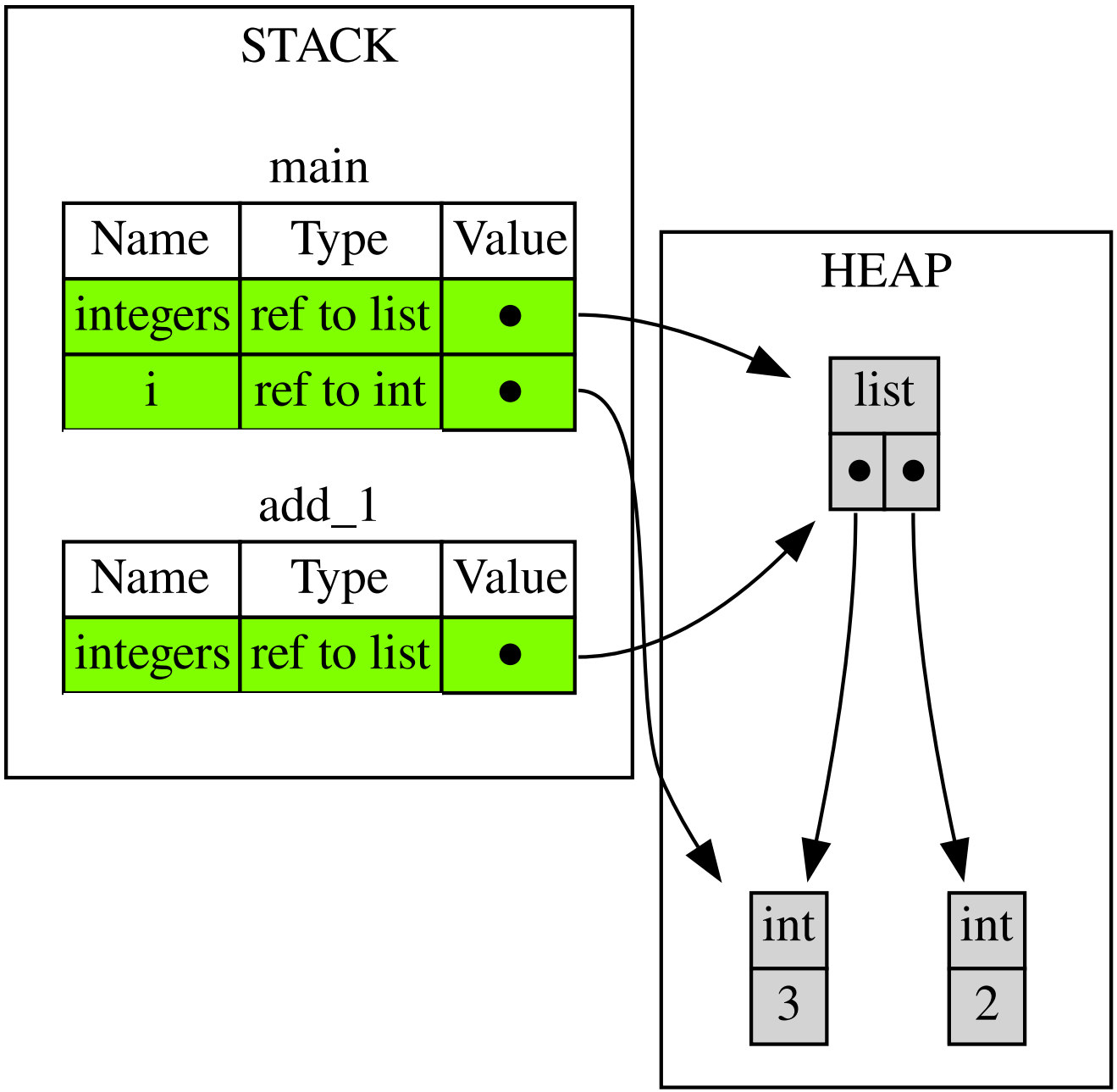


Contribution : ma vie d'enseignant est simplifiée :)

```python
import sys
import easy_tracker

# Initialises le tracker
inferior = sys.argv[1]
tracker_name = "python"
tracker = easy_tracker.init_tracker(tracker_name)

# Démarre l'inférieur
tracker.load_program(inferior)
tracker.start()

# Exécute l'inférieur ligne par ligne
while tracker.get_exit_code() != 0:
    frame = tracker.get_current_frame()
    easy_tracker.draw_stack_heap(frame,
                                 f"at_line_{f.lineno}.svg")
    tracker.step()
```

Contribution : ma vie d'enseignant est simplifiée :)

```
impo
impo

# In
infe
trac
trac

# De
trac
trac

# Ex
whil
```



Contribution : ma vie d'enseignant est simplifiée :)

```
import sys
import easy_tracker

# Initialises le tracker
inferior = sys.argv[1]
tracker_name = "python" if ".py" in (inferior) else "GDB"
tracker = easy_tracker.init_tracker(tracker_name)

# Démarre l'inférieur
tracker.load_program(inferior)
tracker.start()

# Exécute l'inférieur ligne par ligne
while tracker.get_exit_code() != 0:
    frame = tracker.get_current_frame()
    easy_tracker.draw_stack_heap(frame,
                                 f"at_line_{f.lineno}.svg")
    tracker.step()
```

Outils développés et utilisés

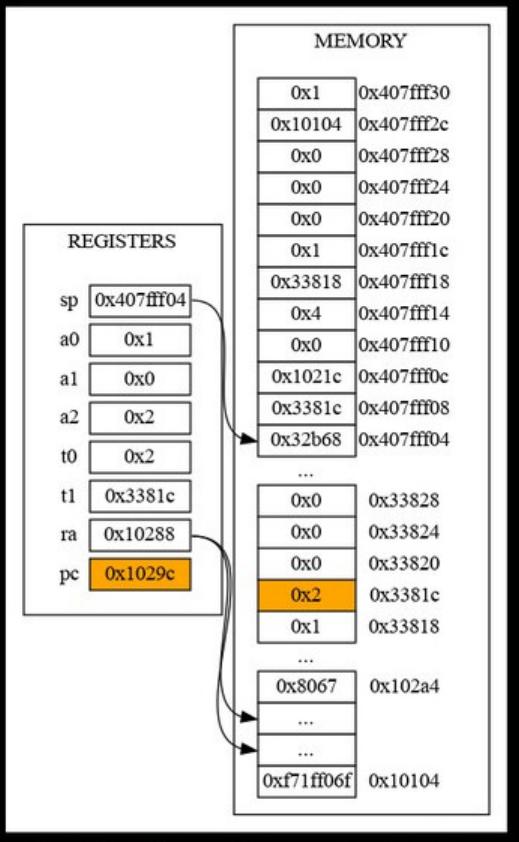
```c
#include <stdlib.h>
#include <inttypes.h>
#include <stdio.h>

uint32_t g = 1;

// Déclaration de la fonction pgcd définie dans fct_pedago.s
extern void f_codee_en_assembleur(uint32_t* a, uint32_t* b, uint32_t c
);
/*
   void f_codee_en_assembleur(uint32_t* a, uint32_t* b, uint32_t c) {
      *b = square(c) + *a;
   }
*/

uint32_t square(uint32_t i) {
   uint32_t res = i * i;
   return res;
}

int main(void) {
   size_t nb_nbs, i;
   printf("Enter a number: \n");
   scanf("%zu", &nb_nbs);
   uint32_t* p = malloc(nb_nbs * sizeof(uint32_t));
   for (i = 0; i < nb_nbs; i++) {
      f_codee_en_assembleur(&g, p + i, i);
      printf("p[i] = %" PRIu32 "\n", *(p + i));
   }
   return 0;
```

inferiors/pedago/pedago.c

```
/* DEBUT DU CONTEXTE
  Fonction :
    f_codee_en_assembleur : non feuille
  Contexte :
    a   : registre a0; pile *(sp+0)
    b   : registre a1; plie *(sp+4)
    c   : registre a2
    ra  : pile *(sp+8)
*/
f_codee_en_assembleur:
    addi sp, sp, -12
    sw   a0, 0(sp)
    sw   a1, 4(sp)
    sw   ra, 8(sp)
f_codee_en_assembleur_fin_prologue:
    /* square(c) */
    mv   a0, a2
    jal  square
```

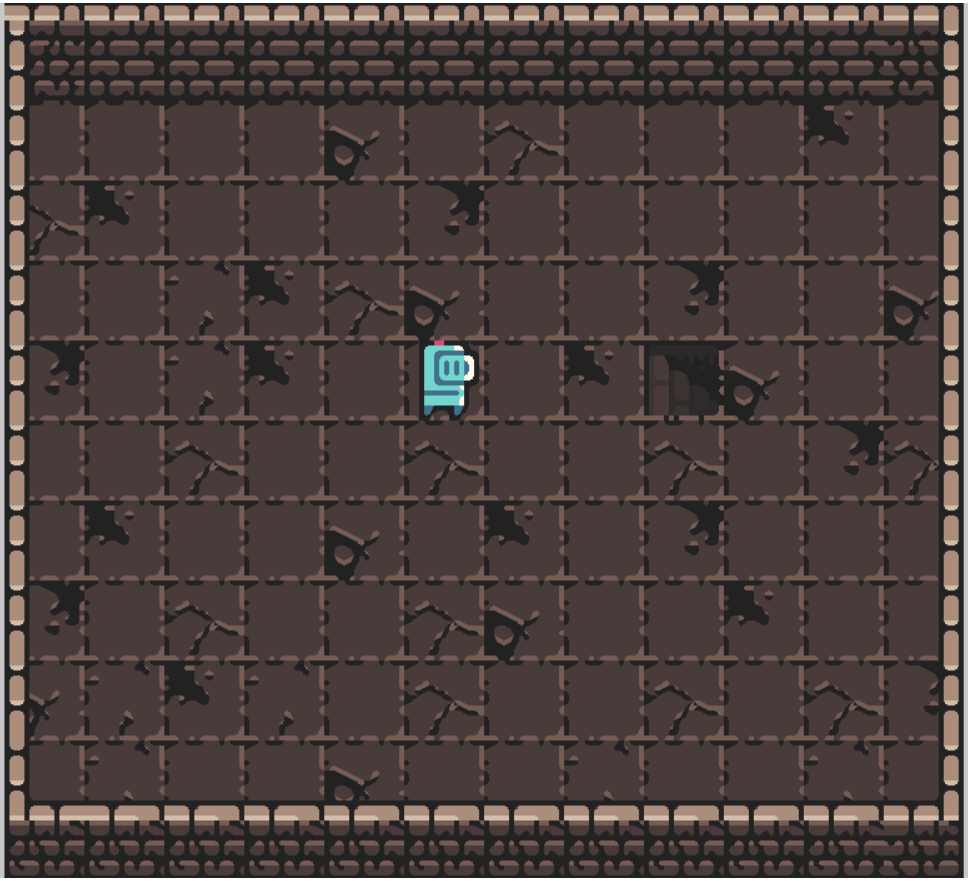
~/boulot/**git-easytracker**/tools/riscv/**sp_visualizator** riscv *2 !4 ?2

kitty +kitten icat visu.png



~/boulot/**git-easytracker**/tools/riscv/**sp_visualizator** riscv *2 !4 ?2

Outils développés et utilisés

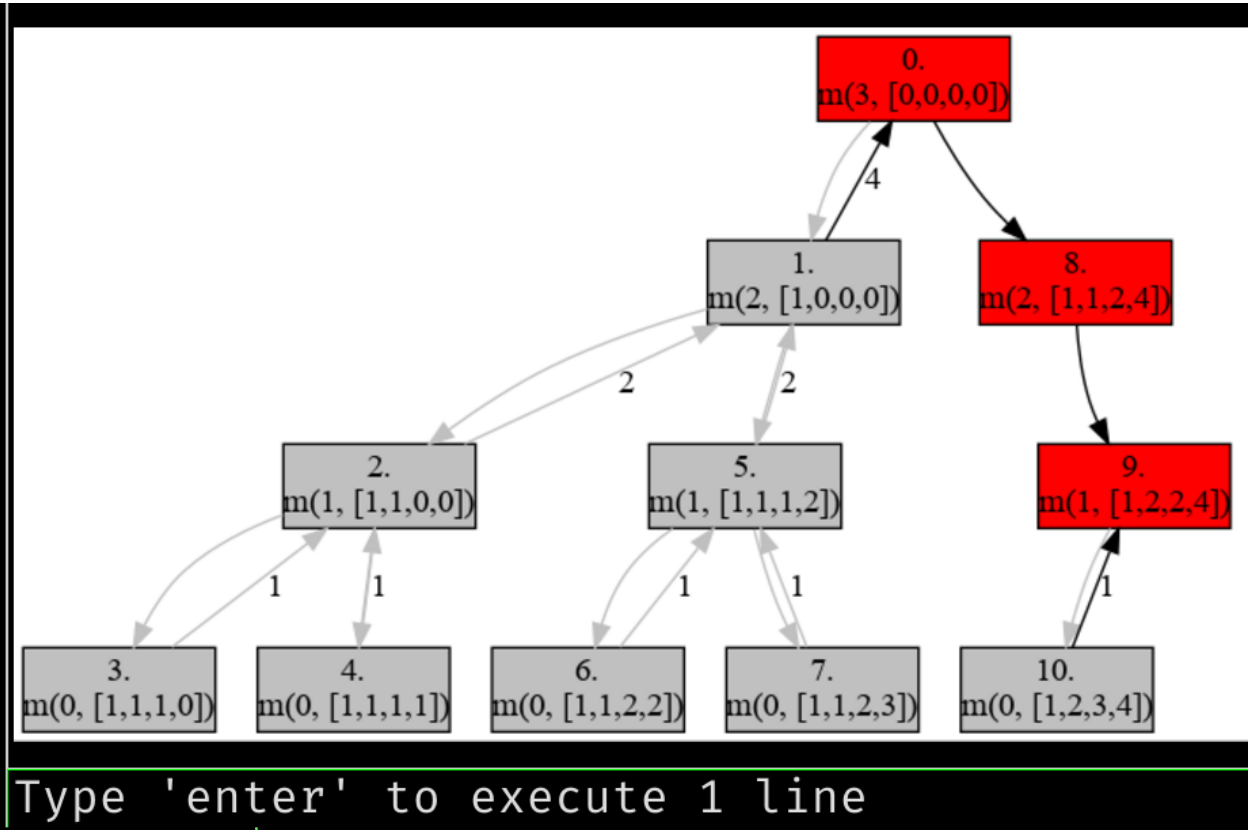


```c
#include <stdio.h>

#include "verify_exit.h"

static int x_character,
y_character, x_exit, y_exit;

void init() {
    x_character = 3;
    y_character = 5;

    x_exit = 8;
    y_exit = 5;
}

void forward(int n) {
    x_character += n;
}

int main(int argc, char **argv) {

    init();

    forward(1);
    forward(1);
```


```
Temporary breakpoint 1, main (argc=1,
argv=0x7fffffffdb68) at level_1.c:21
21          init();

(agdbentures) next
23          forward(1);

(agdbentures) next
24          forward(1);

(agdbentures) next
25          forward(1);

(agdbentures)
```

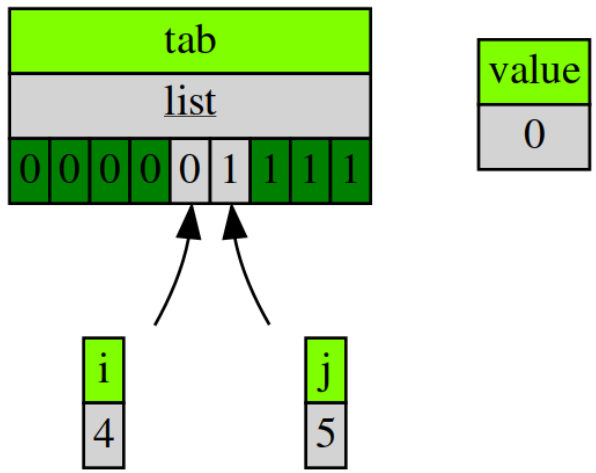
Outils développés et utilisés

```
1  def m(x, y):
2      a = len(y) - 1 - x
3      y[a] += 1
4      if x == 0:
5          return 1
6      m1 = m(x - 1, y)
7      m2 = m(x - 1, y)
8      return m1 + m2
9
10 def main():
11     r = 3
12     b = [0] * (r + 1)
13     print(m(r, b), b)
14
```



Outils développés et utilisés

```
def sort(tab):
    i = 0
    j = len(tab) - 1
    while i != j:
        if tab[i] == 0:
            i = i + 1
        else:
            value = tab[j]
            tab[j] = tab[i]
            tab[i] = value
            j = j - 1
    return tab
def main():
    sort([0, 1, 0, 1, 0,
          1, 0, 1, 0])
```



Perspectives

- Mettre les outils de visualisation **dans les mains des étudiants**
- **Diffuser** la bibliothèque pour améliorer l'API
- Avoir de la visualisation "dynamique"
- Supporter d'autres langages

Bibliographie

Boulay, Benedict du, Tim O'Shea, and John Monk (1981). "The black box inside the glass box: presenting computing concepts to novices". In: International Journal of Man-Machine Studies 14.3, pp. 237–249. ISSN: 0020-7373. DOI: https://doi.org/10.1016/S0020-7373(81)80056-9. URL: https://www.sciencedirect.com/science/article/pii/S0020737381800569.

Fincher, Sally et al. (2020). "Notional Machines in Computing Education: The Education of Attention". In: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education. ITiCSE-WGR '20. Trondheim, Norway: Association for Computing Machinery, 21–50. ISBN: 9781450382939. DOI: 10.1145/3437800.3439202. URL: https://doi.org/10.1145/3437800.3439202.

Guo, Philip J. (2013). "Online Python Tutor: Embeddable Web-Based Program Visualization for Cs Education". In: Proceeding of the 44th ACM Technical Symposium on Computer Science Education. SIGCSE '13. Denver, Colorado, USA: Association for Computing Machinery, 579–584. ISBN: 9781450318686. DOI: 10.1145/2445196.2445368. URL: https://doi.org/10.1145/2445196.2445368.

Notional machines: A Curated Collection (2020). https://notionalmachines.github.io.

Robins, Anthony, Janet Rountree, and Nathan Rountree (2003). "Learning and Teaching Programming: A Review and Discussion". In: Computer Science Education 13.2, pp. 137–172. DOI: 10.1076/csed.13.2.137.14200. eprint: https://doi.org/10.1076/csed.13.2.137.14200. URL: https://doi.org/10.1076/csed.13.2.137.14200.

Sorva, Juha (2013). "Notional Machines and Introductory Programming Education". In: ACM Trans. Comput. Educ. 13.2. DOI: 10.1145/2483710.2483713. URL: https://doi.org/10.1145/2483710.2483713.

Sorva, Juha, Ville Karavirta, and Lauri Malmi (2013). "A Review of Generic Program Visualization Systems for Introductory Programming Education". In: ACM Trans. Comput. Educ. 13.4. DOI: 10.1145/2490822. URL: https://doi.org/10.1145/2490822.

Wagner, Fréderic (2017). tycat: a lightweight Python module to inspect your objects.