« Explicit Tracing »

30 min
5 min

Olivier Goletti & Kim Mens, UCLouvain — 18 mai 2022
Utiliser des stratégies d'instruction explicites dans l'enseignement de la programmation

Cette stratégie vise à enseigner explicitement les étapes systématiques d'exécution d'un code (tracing) via une représentation sur papier de la mémoire et donc le suivi des valeurs intermédiaires. Cette stratégie *« Aide à l'application syntaxique et sémantique lors de apprentissage [des étudiants], en encourageant à tracer ligne-par-ligne et en utilisant une représentation externe de la mémoire à mettre à jour. » [1]*

Avantages :

- Réduit la charge cognitive en évitant aux étudiants de devoir retenir les valeurs intermédiaires et les erreurs de récuperation
- Propose une approche systématique pour dissuader les étudiants de créer leur propre stratégie ou de faire des raccourcis
- Fonctionne comme un debugger

Quand ?

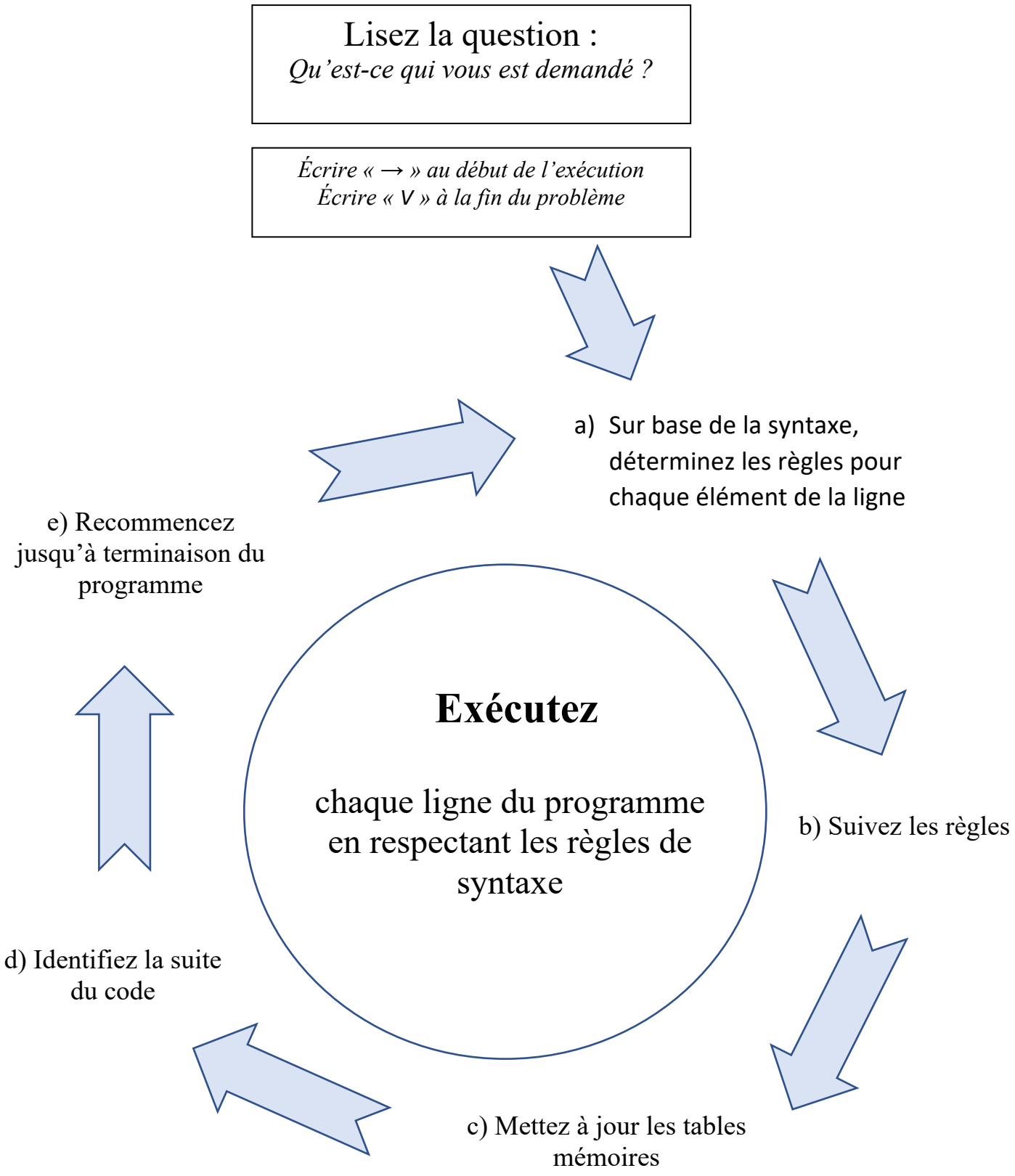
- Dès l'introduction d'un nouveau concept
- Avec des débutants
- Ayant des difficultés à suivre la valeur des variables en mémoire
- Lors d'une erreur de prédiction d'un étudiant sur une exécution de code

/!\ cela ne compense pas un manque de compréhention de la syntaxe et de la sémantique

Comment :

- Expliquer la stratégie
- Exemplifier la stratégie
- Rappeler la syntaxe
- Faire tracer le code par l'étudiant
- Demander à l'étudiant de réfléchir à voix-haute pour identifier les erreurs de compréhension pendant qu'il trace

La Stratégie : Instructions étape par étape



Mémoire externe : repésentation visuelle pour tracer les valeurs des variables

- Séparation des identifiants de variables et de leurs valeurs pour différencier :
  - Identifiants des variables et strings
  - Identifiants des fonctions et variables locales
  - L'importance de la casse des identifiants des variables
- La portée des variables
  - Environnement global vs. environnement d'appel d'une fonction
- Exemple de visualisation d'exécution sur http://pythontutor.com

Pour chaque programme ou appel de fonction

① Créer une **nouvelle** table

② Nommer la table

représentant soit l'environnement global soit l'environnement d'appel d'une fonction
en cas de de l'environnement global, utiliser GLOBAL comme nom de table ;
sinon nom de table = identifiant de la fonction

Pour chaque nouvelle variable

③ Créer une **nouvelle** ligne dans la table

avec l'identifant de la varibale et sa valeur

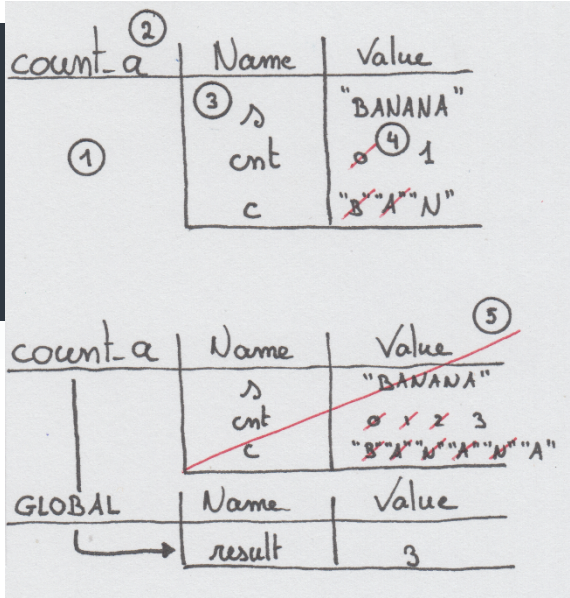
Quand une variable est mise-à-jour

④ Trouver la variable, barrer la valeur précédente et écrire la nouvelle

Quand la fonction est achevée

⑤ Écrire la valeur de retour, barrer l'entièreté de la table

```
def count_a(s):
    cnt = 0
    for c in s:
        if c == "A":
            cnt += 1
    return cnt

result = count_a("BANANA")
```

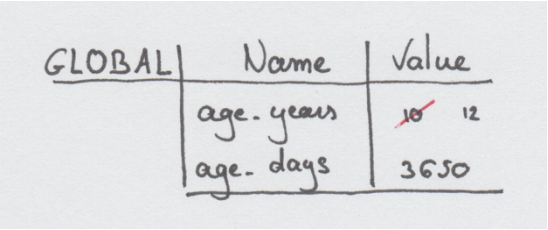


Exemples pour des types de valeurs différents

Types primitifs :

Les valeurs sont barrées lorsqu'elles sont modifiées afin qu'un historique des valeurs précédentes puisse être conservé.

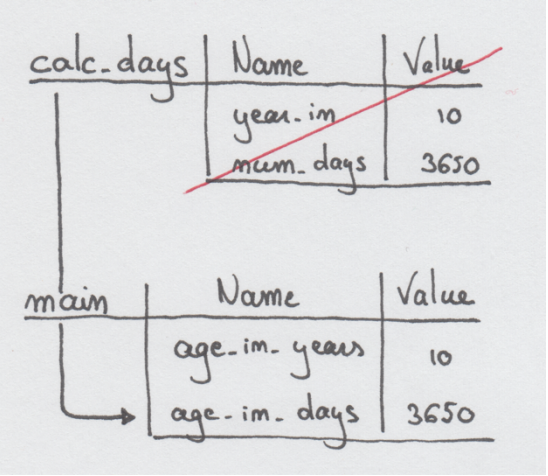
```
age_years = 10
age_days = age_years * 365
age_years = 12
```



Appels de fonction :

Schématisés sous forme de tables individuelles démontrent à la fois le concept de stack et de portée. Lorsque la fonction se termine, barrez la table qui sera devenue hors portée et indiquez les valeurs de retour par une flèche.

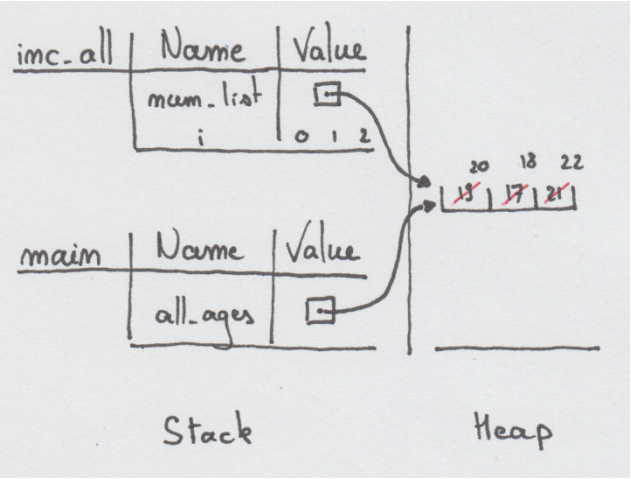
```
def calc_days(years_in):
    num_days = years_in * 365
    return num_days

def main():
    age_in_years = 10

age_in_days = calc_days(age_in_years)
main()
```



Mémoire dynamique :

Les flèches représentant des pointeurs vers une adresse mémoire permettent d'expliquer le concept de passer des références comme arguments.

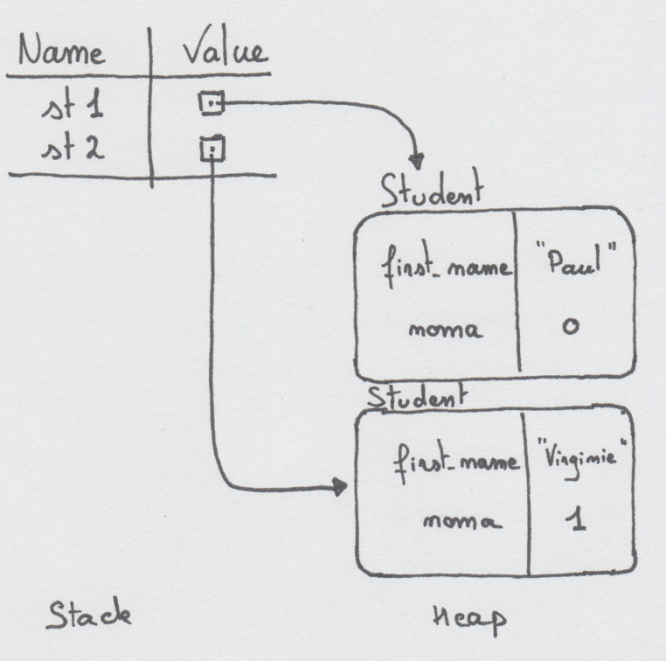
```
def inc_all(num_list):
    for i in
range(len(num_list)):
        num_list[i] += 1

def main():
    all_ages = [19, 17, 21]
    inc_all(all_ages)

main()
```



Objets :

Stockés dynamiquement sur le heap, les instances des objets sont représentées par des rectangles arrondis.

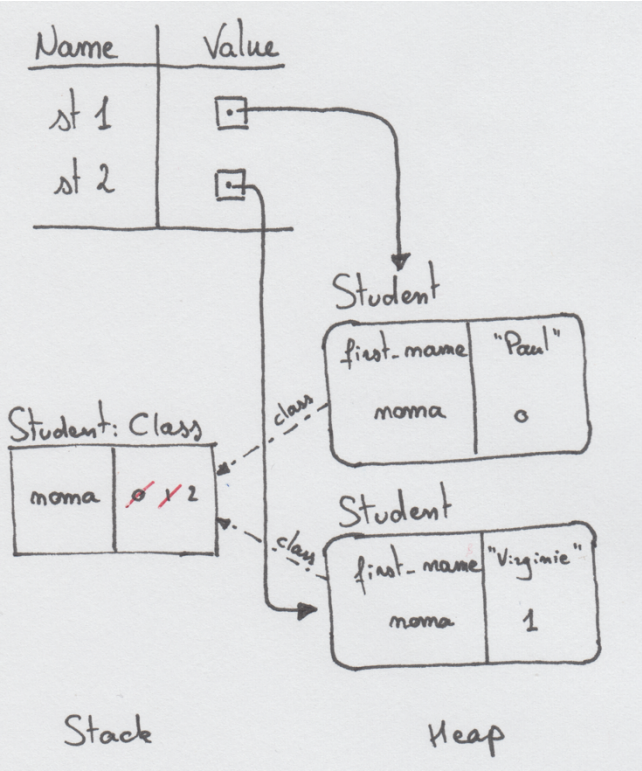
```
st1 = Student("Paul")
st1.noma = 0
st2 = Student("Virginie")
st2.noma = 1
```



Classes :

Commes les fonctions, les classes sont définies statiquement et on les représente par des rectangles. Dans ces rectangles on ne stocke que les valeurs des variables de classe. Cette notation ne doit être utilisée que s'il y a des variables de classe.

```
class Student:
    noma = 0

    def __init__(self,
first_name):
        self.first_name =
first_name
        self.noma = Student.noma
        Student.noma += 1

st1 = Student("Paul")
st2 = Student("Virginie")
```



Observations intéressantes

- Une compréhension incomplète de la sémantique peut être une source de déviation du traçage ligne-par-ligne (e.g. x++, 5 !, break, continue, pass, …).

- Créer le réflexe de faire des tables mémoires lorsqu'il y a plusieurs variables à mettre à jour. Moins systématique pour les chaînes de caractères (plus long à écrire).

- « Au début les élèves n'ont pas de méthodologie pour tracer un code. Ils essaient de retenir les variables. Alors qu'avec le tableau ils se posent, écrivent bien chaque expression et calculent chaque expression séparément, je pense que ça les a aidé ». [3]

- « Avec tout ce qui est boucle et iterateurs, ça marche bien. Quand je leur dis "tu vois la méthode avec le tableau?" tu peux la réutiliser ». [3]

- « Les étudiants ont bien aimé la stratégie, parce qu'elle reste très simple et très concrète. On peut leur montrer avec un exemple qui dure une minute que ça fonctionne et ça leur permet de trouver là où il y a une erreur ». [3]

- « L'occasion s'est très vite présentée avec un étudiant qui disait, oui mais si on fait ça est-ce que ça marche ou pas ? Ben j'ai dit : Ok ! moi j'ai une technique à vous montrer. C'était très simple à mettre en place ». [3]

- « C'était vraiment libérer de la charge mentale des étudiants et qu'ils se concentrent sur l'exécution. Je vois que quand ils le font dans leur tête, ils ont du mal et alors je leur dis : "C'est pas le but, Il faut tout écrire !" Ca leur permet de se vider la tête, avec 4-5 variables, ils prennent des risques ». [3]

Références

1. Benjamin Xie, Greg L. Nelson and Amy J. Ko. 2018. An Explicit Strategy to Scaffold Novice Program Tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 344-349.
2. Toby Dragon and Paul E. Dickson. 2021. A Memory Diagram for All Seasons. *In ITiCSE '21, Virtual Event, Germany*. 150-156.
3. Olivier Goletti, Kim Mens and Felienne Hermans. 2021. Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course. *In ITiCSE '21, Virtual Event, Germany*.

5 min

Mise en situation

Utilisez la stratégie du *explicit tracing* pour tracer le code suivant, pour x = 3.
[Extrait de la mission 1 : Q* Factorial]

```
def fact(x):
    result = 1
    if x != 0 :
        for i in range(1,x+1) :
            result = i * result
    return result

fact(3)
```