Quelques notions de didactique de l'informatique pour les mathématiciens

Bruno Mermet

Avril 2021

Informatique : kesako ?

- Définition

  L'informatique est la science du traitement de l'information

- Concepts clés
  - Science
  - Information → données
  - Traitement :
    - Principe → algorithme
    - Description formelle → programme, dans un langage
    - Exécution → processus, sur une machine

Informatique et Mathématiques

- L’informatique a besoin des mathématiques
  - Informatique est une science
    - raisonnement formel sur des modèles formels
    - Les mathématiques sont un outil pour l’informatique
  - Informatique = outil pour d’autres sciences utilisant des modèles mathématiques
    - Des notions de mathématiques sont nécessaires pour l’écriture de certaines applications
- Les mathématiques ont besoin de l’informatique
  - Démonstration (théorème des 4 couleurs conjecture en 1852, démontré en 1976 sur 1478 cas de base)
  - Simulation, calcul formel, visualisation, etc.
- Lien
  - Informatique = Mathématiques ?
  - Informatique $\subset$ Mathématiques ?
  - Mathématiques $\subset$ Informatique ?

La temporalité

- En mathématique, une propriété peut-être
  - Vrai (démontrée comme telle, ou admise)
  - Fausse (démontrée comme telle, ou admise)
  - (Pas encore démontrée ni admise)

  Pas de notion de temporalité !
- En informatique, la valeur d’une propriété peut évoluer

Notion d’état

- Rappels
  - En mathématiques, le monde est immuable
  - En informatique, le monde change
- Définition

  on appelle *état du monde* l’ensemble des valeurs des différentes caractéristiques du monde à un instant donné

Changement d’état et affectation

- Changement d’état

  En informatique, on va décrire comment on passe d’un état à un autre, autrement dit comment une caractéristique va changer de valeur

  - Exemple
    - Aujourd’hui, je suis au Havre, la caractéristique « position » vaut donc « Havre »
    - Demain je serai à Caen, la caractéristique « position » vaudra alors « Caen »
- Affectation
  - il faut donc que j’aie un moyen de décrire que la caractéristique « position » change ; que je lui **affecte** une nouvelle valeur

    position ← Caen
  - L’affectation est donc un changement d’état élémentaire

En première conclusion : spécificités de l'informatique

- L'**affectation** (liée à la notion de changement d'état) est une notion spécifique à l'informatique
- Un changement d'état prend un certain temps (extrêmement court, mais non nul)

→ notion d'**efficacité**

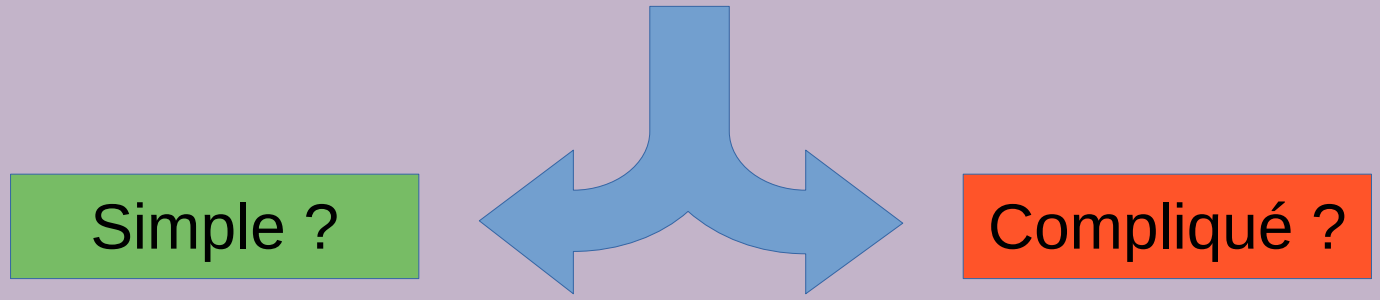
> Equality is a simple concept that five-year-old children understand.
> Assignement is a complicated concept that university students find difficult.
> Equality obeys simple algebraic law; assignement doesn't.
>
> L. Lamport

Exemple de programme

```
bleue = 2
print(bleue)
rouge = 1 + bleue
print(rouge)
rouge = bleue + rouge
```

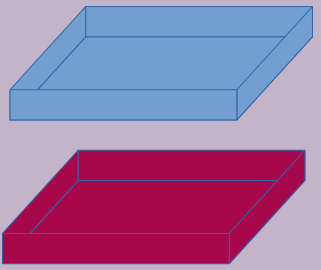
Exemple de programme

```
bleue = 2
print(bleue)
rouge = 1 + bleue
print(rouge)
rouge = bleue + rouge
```



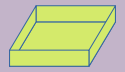
Vision débranchée de l'affectation

- On dispose de 2 boîtes ne pouvant contenir qu'un post-it®:
  - 1 bleue
  - 1 rouge



Les valeurs sont des nombres écrits sur des post-it®

Notre main peut contenir 1 seul post-it®



Stocker un post-it dans une boîte « écrase » l'ancien post-it

- On peut alors écrire le *programme* suivant :
  - `Stocker 2 dans la boîte bleue`
  - `Afficher la valeur de la boîte bleue`
  - `Stocker l'ajout de 1 à la valeur de la boîte bleue dans la boîte rouge`
  - `Afficher la valeur de la boîte rouge`
  - `Stocker l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge dans la boîte rouge`

Transformer l'écriture du programme (1)

- On avait :
  - `Stocker 2 dans la boîte bleue`
  - `Afficher la valeur de la boîte bleue`
  - `Stocker l'ajout de 1 à la valeur de la boîte bleue dans la boîte rouge`
  - `Afficher la valeur de la boîte rouge`
  - `Stocker l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge dans la boîte rouge`
- On propose de remplacer `stocker` *x* `dans` *v* par *x* ← *v* :
  - `boîte bleue ← 2`
  - `Afficher la valeur de la boîte bleue`
  - `boîte rouge ← l'ajout de 1 à la valeur de la boîte bleue`
  - `Afficher la valeur de la boîte rouge`
  - `boîte rouge ← l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge`

Transformer l'écriture du programme (1)

- On avait :
  - `Stocker 2 dans la boîte bleue`
  - `Afficher la valeur de la boîte bleue`
  - `Stocker l'ajout de 1 à la valeur de la boîte bleue dans la boîte rouge`
  - `Afficher la valeur de la boîte rouge`
  - `Stocker l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge dans la boîte rouge`
- On propose de remplacer `stocker` *x* `dans` *v* par *x* ← *v* :
  - `boîte bleue ← 2`
  - `Afficher la valeur de la boîte bleue`
  - `boîte rouge ← l'ajout de 1 à la valeur de la boîte bleue`
  - `Afficher la valeur de la boîte rouge`
  - `boîte rouge ← l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge`

Introduction d'une notation informatique

Transformer l'écriture du programme (2)

- On avait :
  - `boîte bleue ← 2`
  - `Afficher la valeur de la boîte bleue`
  - `boîte rouge ← l'ajout de 1 à la valeur de la boîte bleue`
  - `Afficher la valeur de la boîte rouge`
  - `boîte rouge ← l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge`
- On propose de remplacer `boîte` *x* par *x* :
  - `bleue ← 2`
  - `Afficher la valeur de la bleue`
  - `rouge ← l'ajout de 1 à la valeur de la bleue`
  - `Afficher la valeur de la rouge`
  - `rouge ← l'ajout de la valeur de la bleue à la valeur de la rouge`

Transformer l'écriture du programme (2)

- On avait :
  - `boîte bleue ← 2`
  - `Afficher la valeur de la boîte bleue`
  - `boîte rouge ← l'ajout de 1 à la valeur de la boîte bleue`
  - `Afficher la valeur de la boîte rouge`
  - `boîte rouge ← l'ajout de la valeur de la boîte bleue à la valeur de la boîte rouge`
- On propose de remplacer `boîte` *x* par *x* :
  - `bleue ← 2`
  - `Afficher la valeur de la bleue`
  - `rouge ← l'ajout de 1 à la valeur de la bleue`
  - `Afficher la valeur de la rouge`
  - `rouge ← l'ajout de la valeur de la bleue à la valeur de la rouge`

Perte de la notion de « contenant »

Transformer l'écriture du programme (3)

- On avait :
  - `bleue ← 2`
  - `Afficher la valeur de la bleue`
  - `rouge ← l'ajout de 1 à la valeur de la bleue`
  - `Afficher la valeur de la rouge`
  - `rouge ← l'ajout de la valeur de la bleue à la valeur de la rouge`
- On propose de remplacer `la valeur de la` *x* par *x* :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← l'ajout de 1 à bleue`
  - `Afficher rouge`
  - `rouge ← l'ajout de bleue à rouge`

Transformer l'écriture du programme (3)

- On avait :
  - `bleue ← 2`
  - `Afficher la valeur de la bleue`
  - `rouge ← l'ajout de 1 à la valeur de la bleue`
  - `Afficher la valeur de la rouge`
  - `rouge ← l'ajout de la valeur de la bleue à la valeur de la rouge`
- On propose de remplacer `la valeur de la` *x* par *x* :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← l'ajout de 1 à bleue`
  - `Afficher rouge`
  - `rouge ← l'ajout de bleue à rouge`

Confusion
Valeur / contenant

Transformer l'écriture du programme (4)

- On avait :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← l'ajout de 1 à bleue`
  - `Afficher rouge`
  - `rouge ← l'ajout de bleue à rouge`
- On propose de remplacer `l'ajout de` *x* `à` *y* par *x* + *y* :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← 1 + bleue`
  - `Afficher rouge`
  - `rouge ← bleue + rouge`

Transformer l'écriture du programme (4)

- On avait :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← l'ajout de 1 à bleue`
  - `Afficher rouge`
  - `rouge ← l'ajout de bleue à rouge`
- On propose de remplacer `l'ajout de` *x* `à` *y* par *x* + *y* :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← 1 + bleue`
  - `Afficher rouge`
  - `rouge ← bleue + rouge`

Introduction d'une notation mathématique

Transformer l’écriture du programme (5)

- On avait :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← 1 + bleue`
  - `Afficher rouge`
  - `rouge ← bleue + rouge`
- On propose de remplacer `x ← y` par *x = y* :
  - `bleue = 2`
  - `Afficher bleue`
  - `rouge = 1 + bleue`
  - `Afficher rouge`
  - `rouge = bleue + rouge`

Transformer l'écriture du programme (5)

- On avait :
  - `bleue ← 2`
  - `Afficher bleue`
  - `rouge ← 1 + bleue`
  - `Afficher rouge`
  - `rouge ← bleue + rouge`
- On propose de remplacer `x ← y` par *x = y* :
  - `bleue = 2`
  - `Afficher bleue`
  - `rouge = 1 + bleue`
  - `Afficher rouge`
  - `rouge = bleue + rouge`

Modification de sens d'un symbole bien connu

Un programme en résumé

Introduction d'une notation informatique

+

Perte de la notion de « contenant »

+

Confusion
Valeur / contenant

+

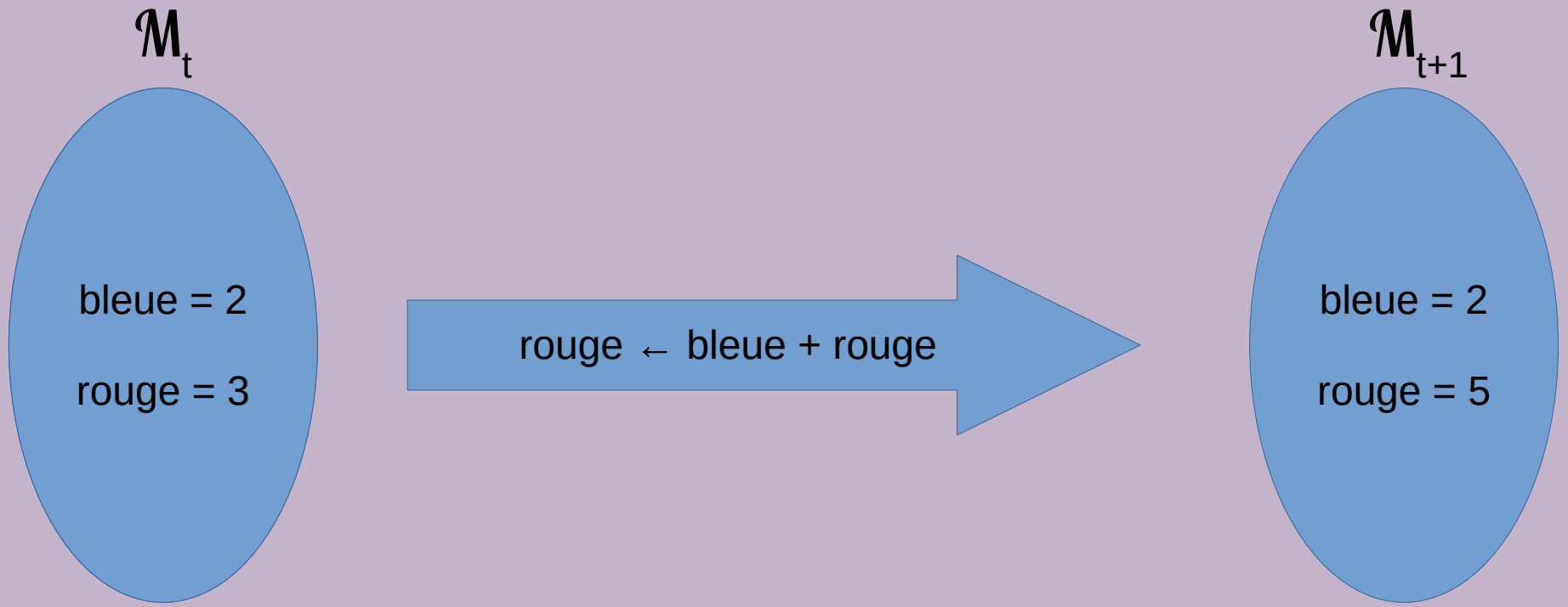
Introduction d'une notation mathématique

+

Modification de sens d'un symbole bien connu

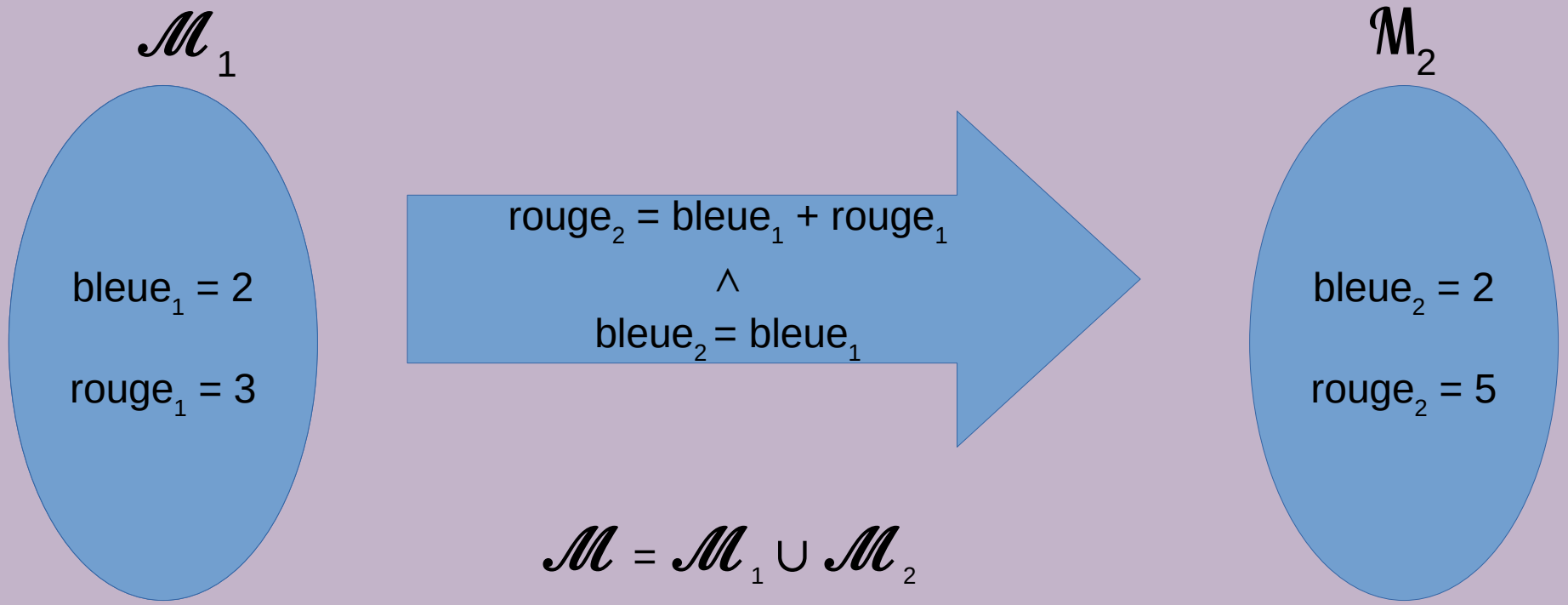
Vision mathématique de l'affectation (1)

- En informatique, l'affectation fait évoluer le monde



Vision mathématique de l'affectation (1)

- En mathématiques, l'affectation établit une relation entre 2 sous-mondes



Visualiser l’affectation

- PythonTutor est un site web qui permet d’exécuter pas à pas un programme écrit en Python en affichant au fur et à mesure la valeur des variables
- Préambule rapide sur Python
  - L’affectation se note « = »
  - Les variables ne se déclarent pas
  - Les variables ne sont pas typées
  - On utilise `print(v)` pour afficher la valeur *v*

Saisie du programme

Write code in Python 3.6

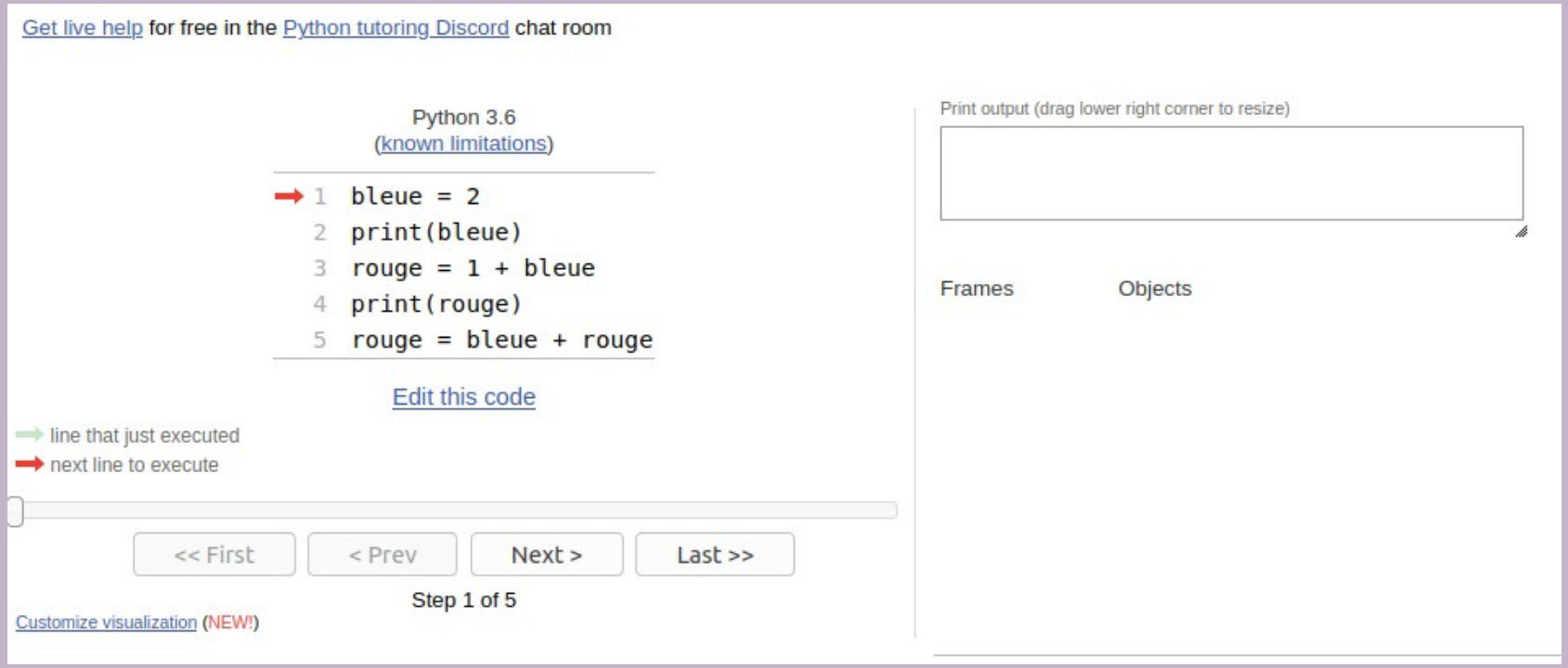
```
1  bleue = 2
2  print(bleue)
3  rouge = 1 + bleue
4  print(rouge)
5  rouge = bleue + rouge
```

Visualize Execution Live Programming Mode

hide exited frames [default] inline primitives, don't nest objects [default]

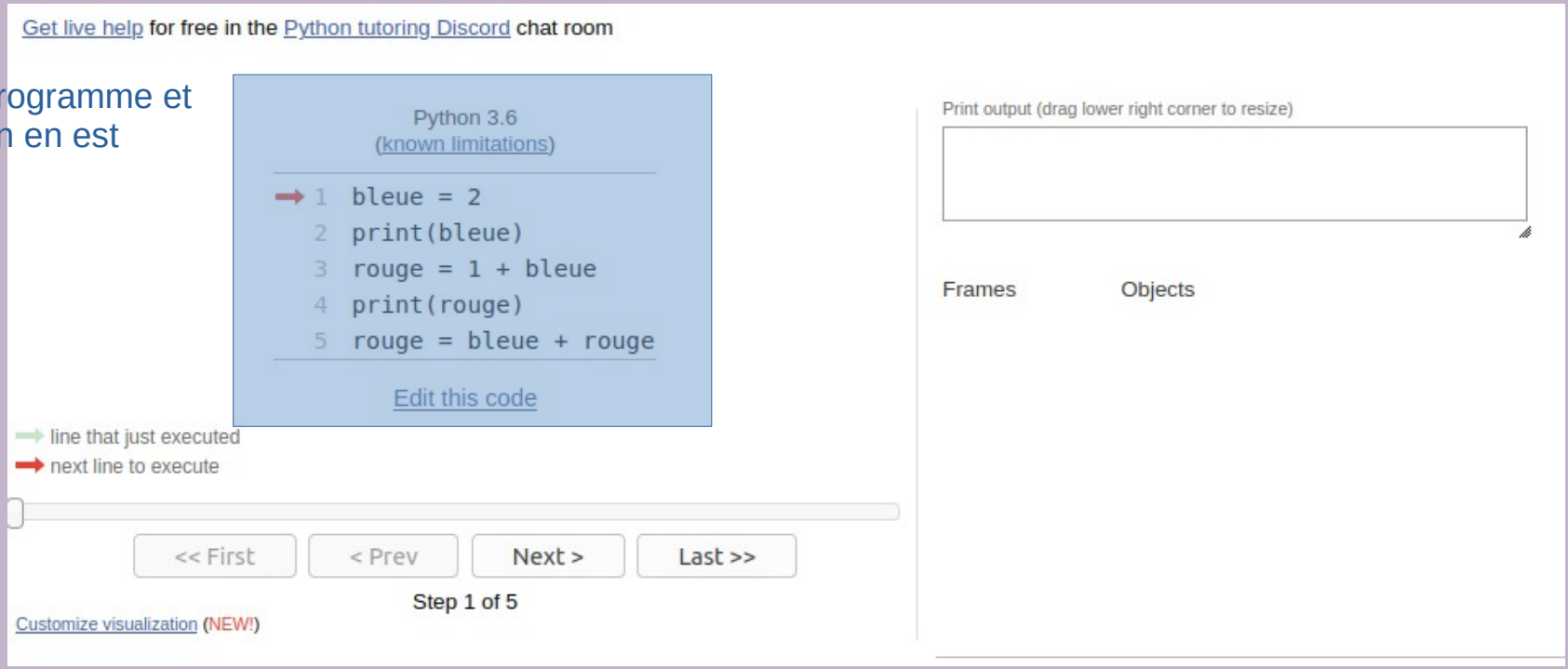
draw pointers as arrows [default]

Exécution pas à pas (1)



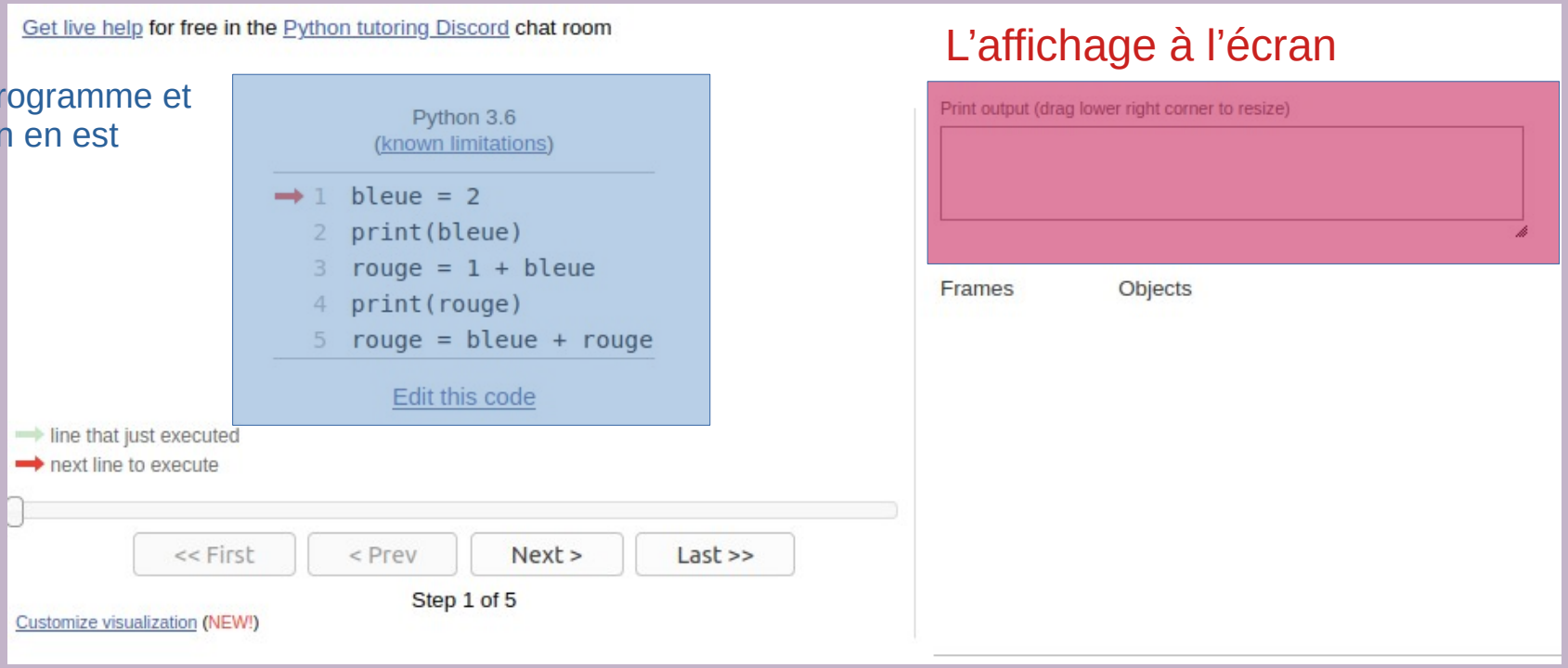
Exécution pas à pas (1)

Le programme et où on en est

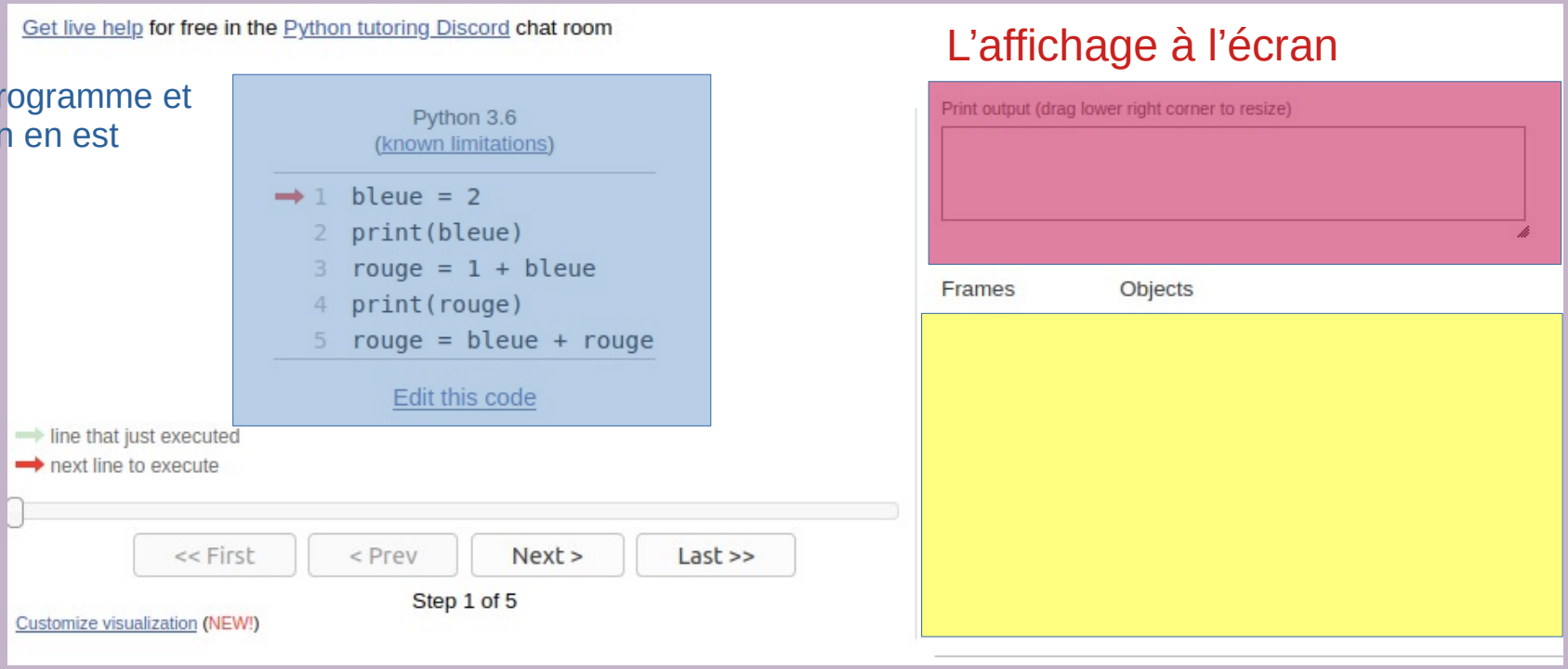


Exécution pas à pas (1)

Le programme et où on en est



Exécution pas à pas (1)

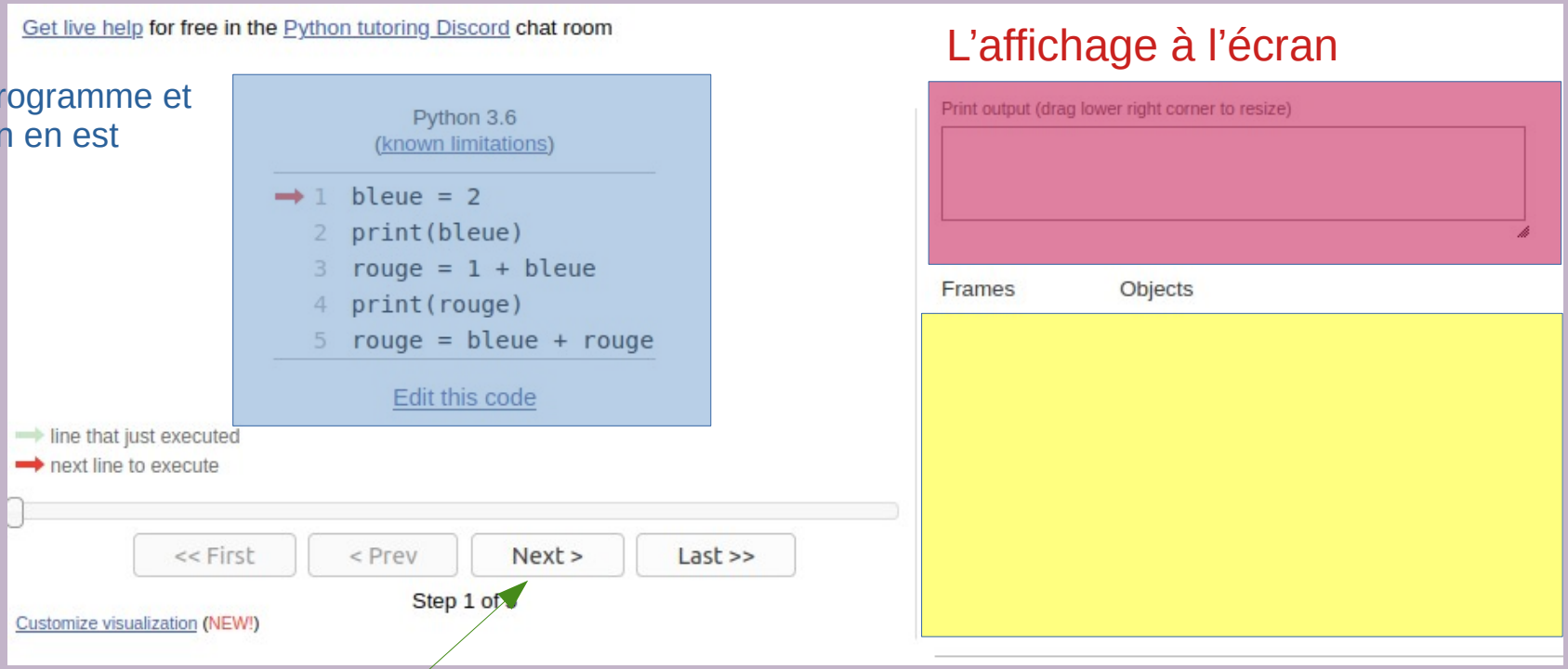


Le programme et où on en est

L'affichage à l'écran

L'état du programme

Exécution pas à pas (1)



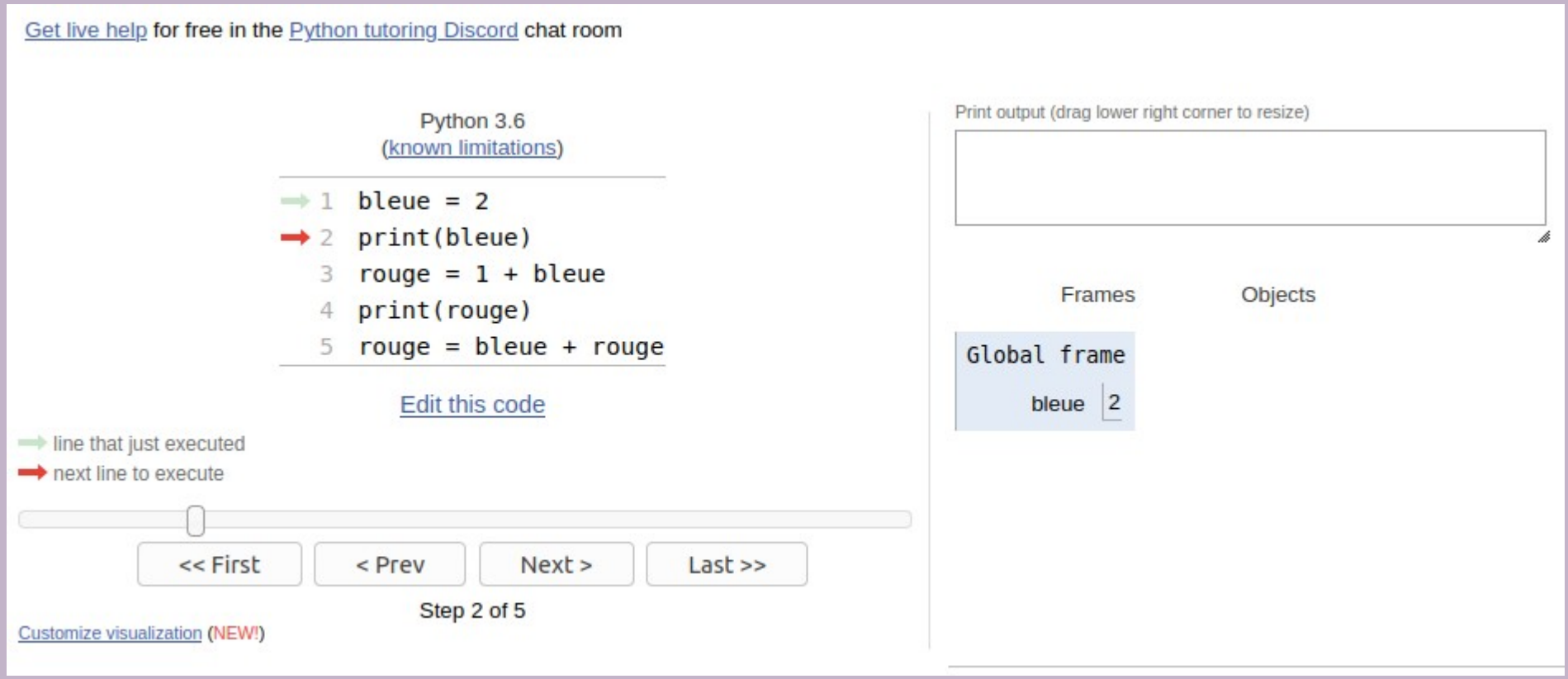
Le programme et où on en est

L'affichage à l'écran

L'état du programme

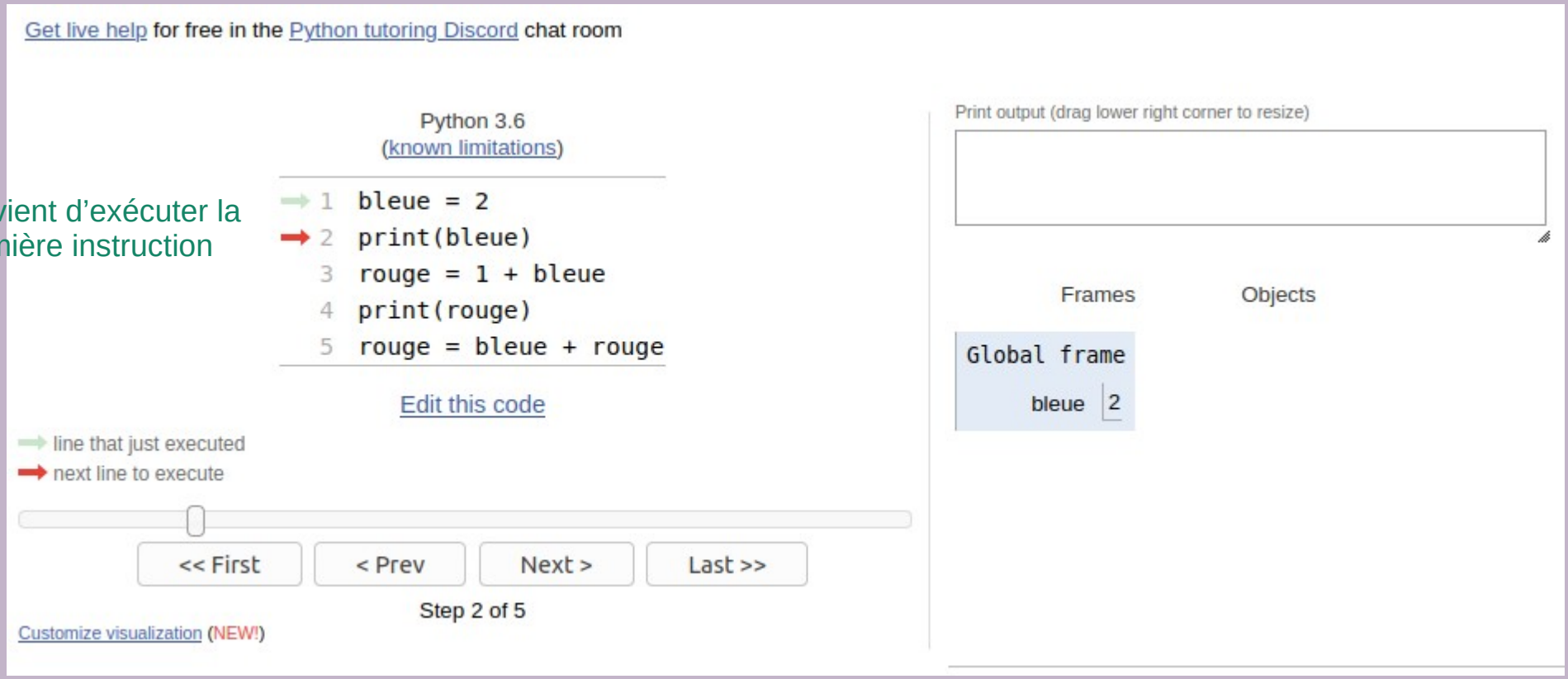
Bouton pour exécuter la prochaine instruction

Exécution pas à pas (2)

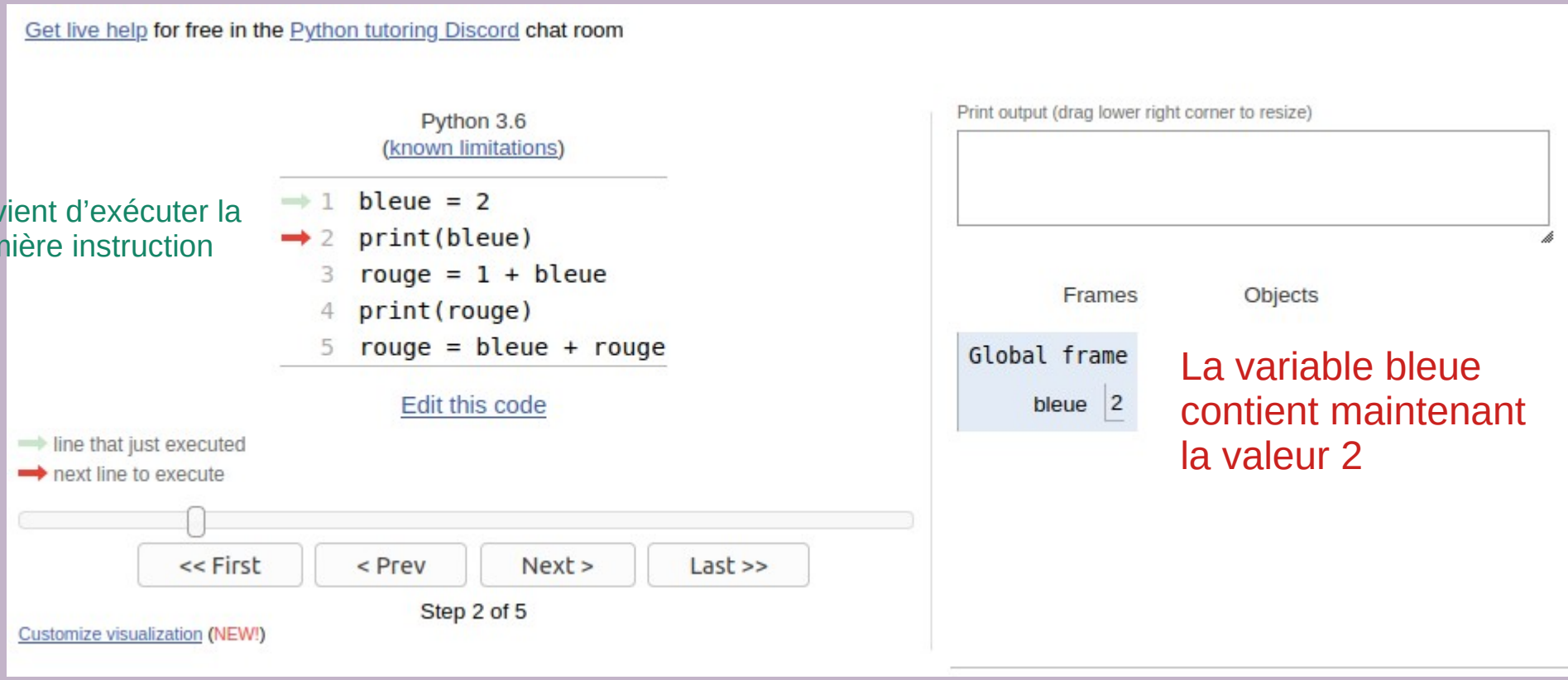


Exécution pas à pas (2)

On vient d'exécuter la première instruction



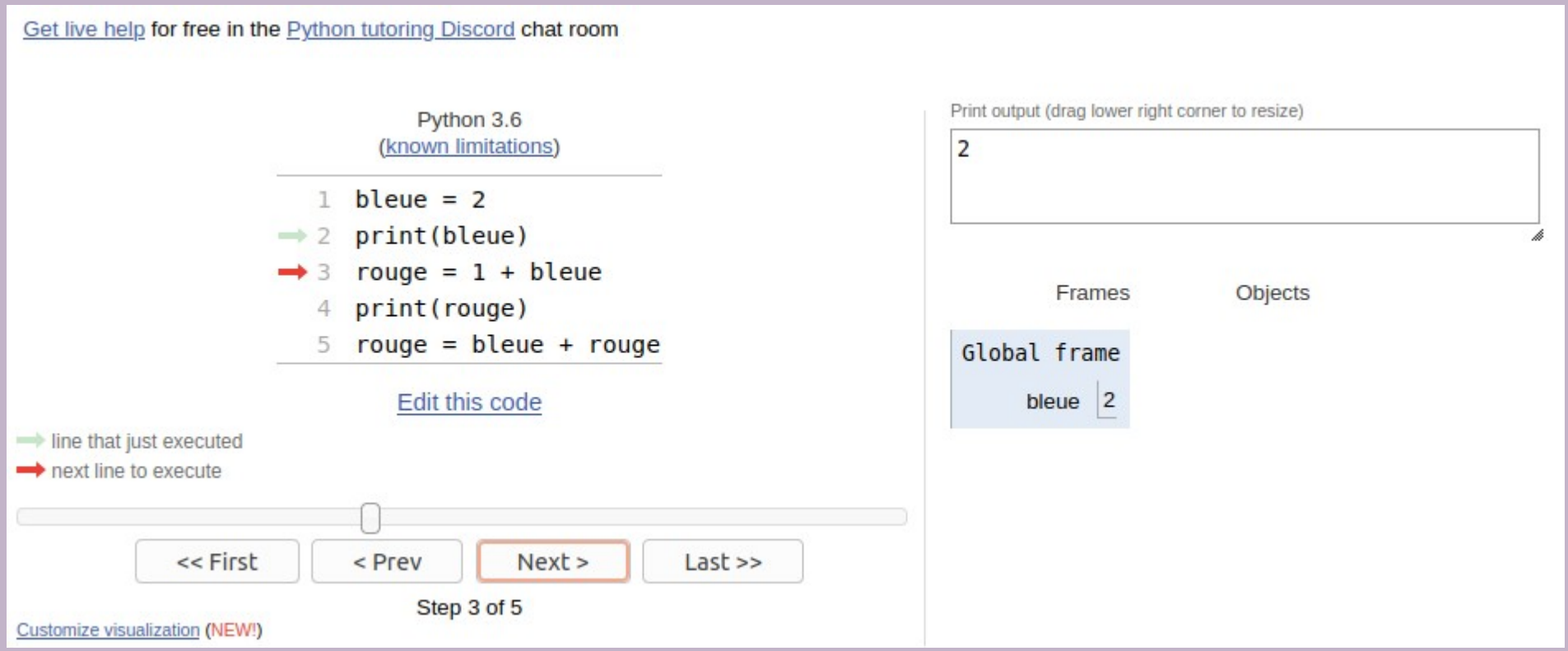
Exécution pas à pas (2)



On vient d'exécuter la première instruction

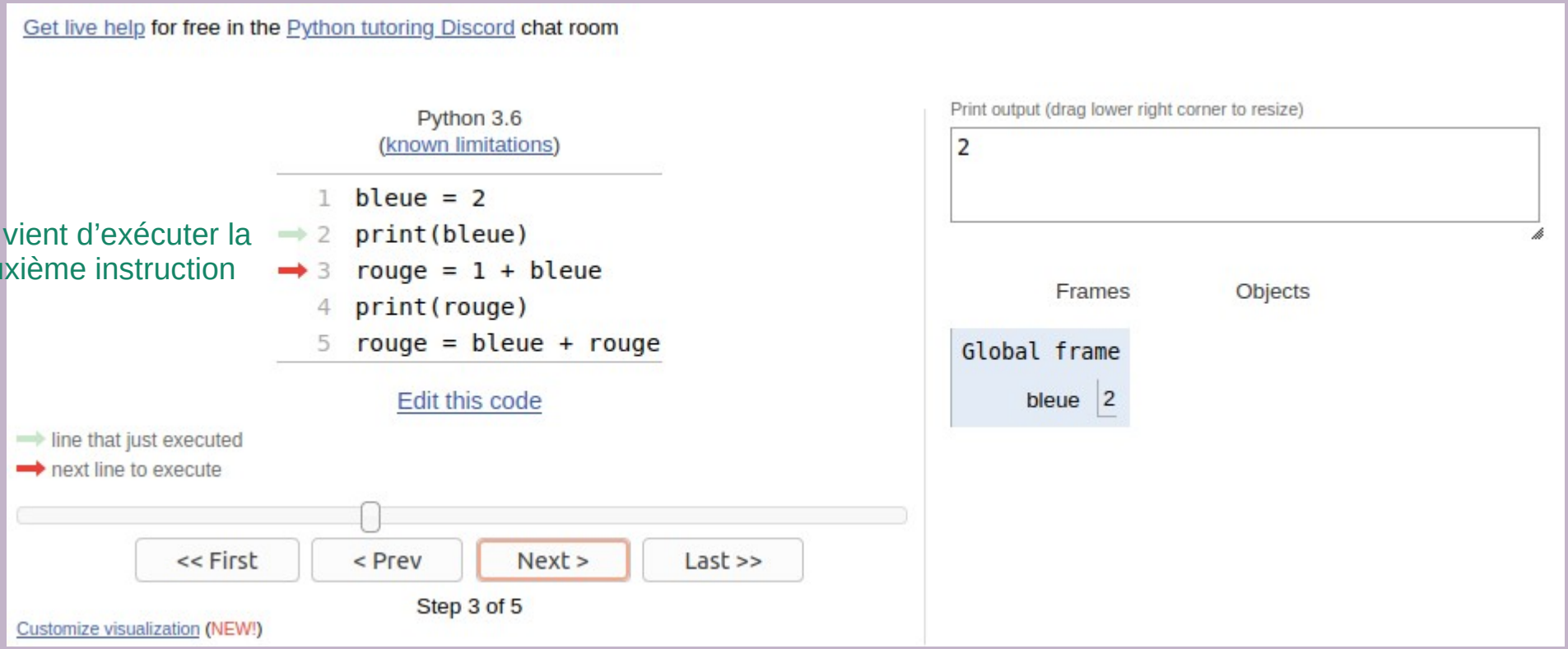
La variable bleue contient maintenant la valeur 2

Exécution pas à pas (3)

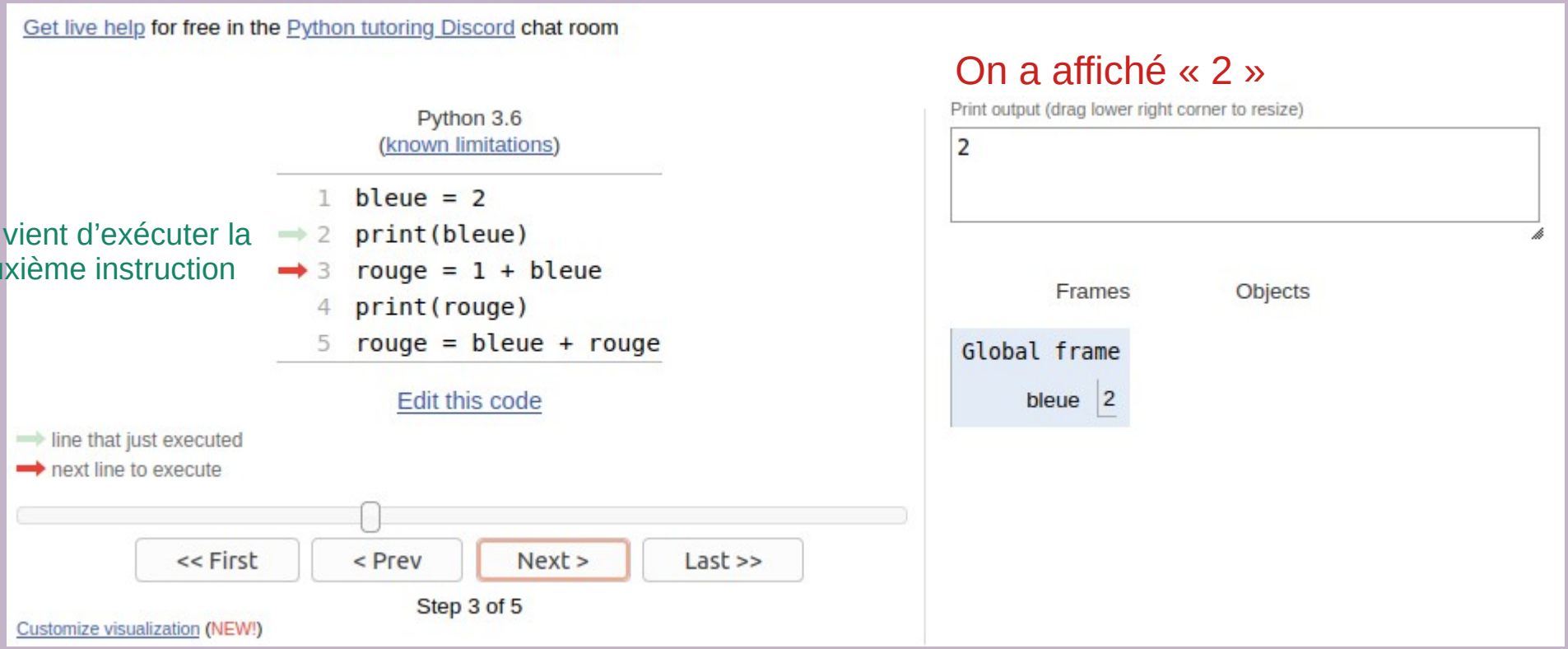


Exécution pas à pas (3)

On vient d'exécuter la deuxième instruction



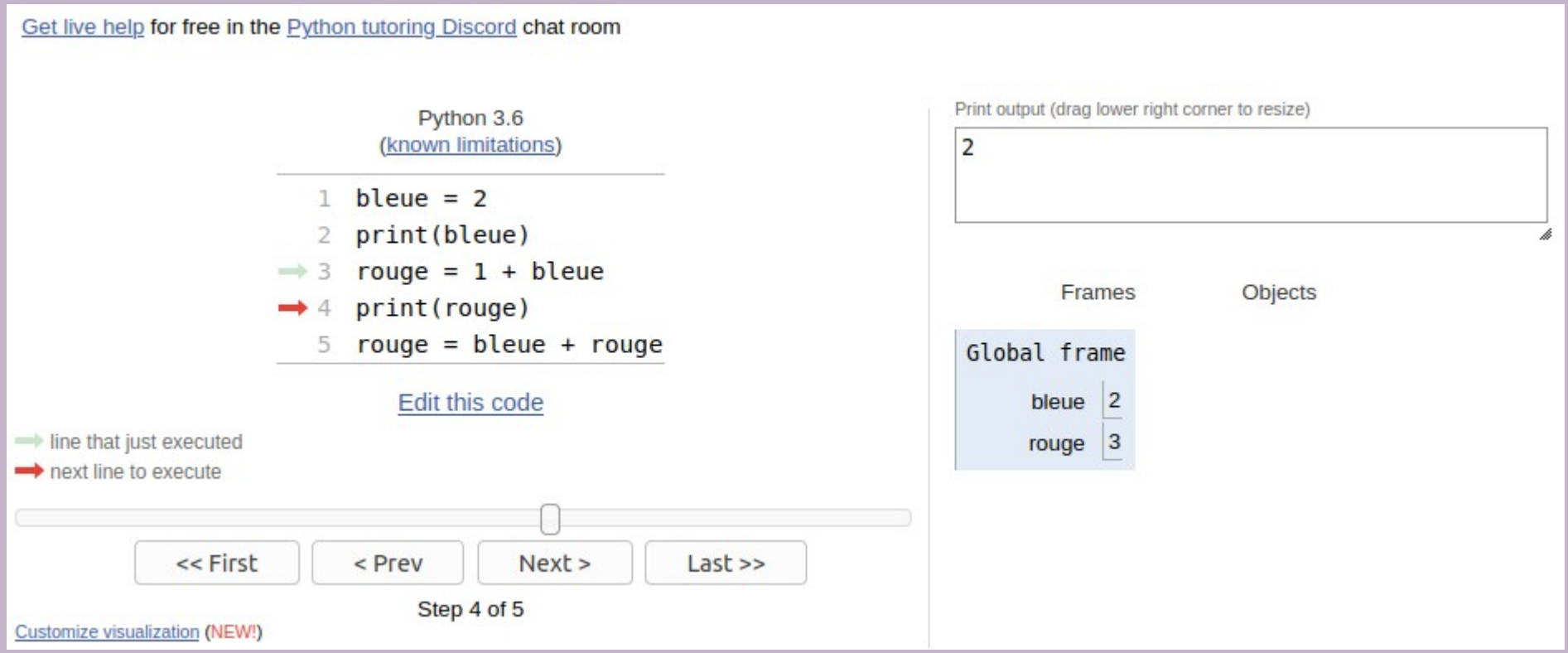
Exécution pas à pas (3)



On vient d'exécuter la deuxième instruction

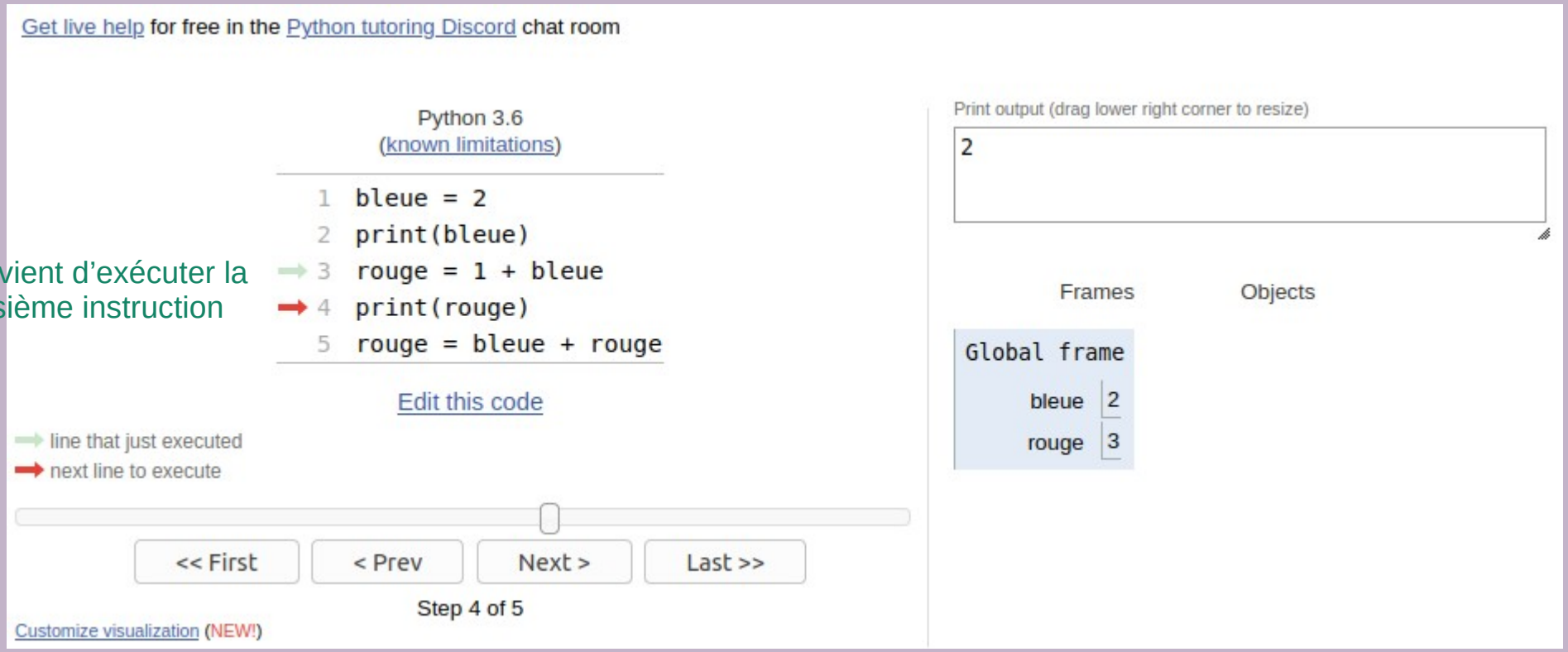
On a affiché « 2 »

Exécution pas à pas (4)

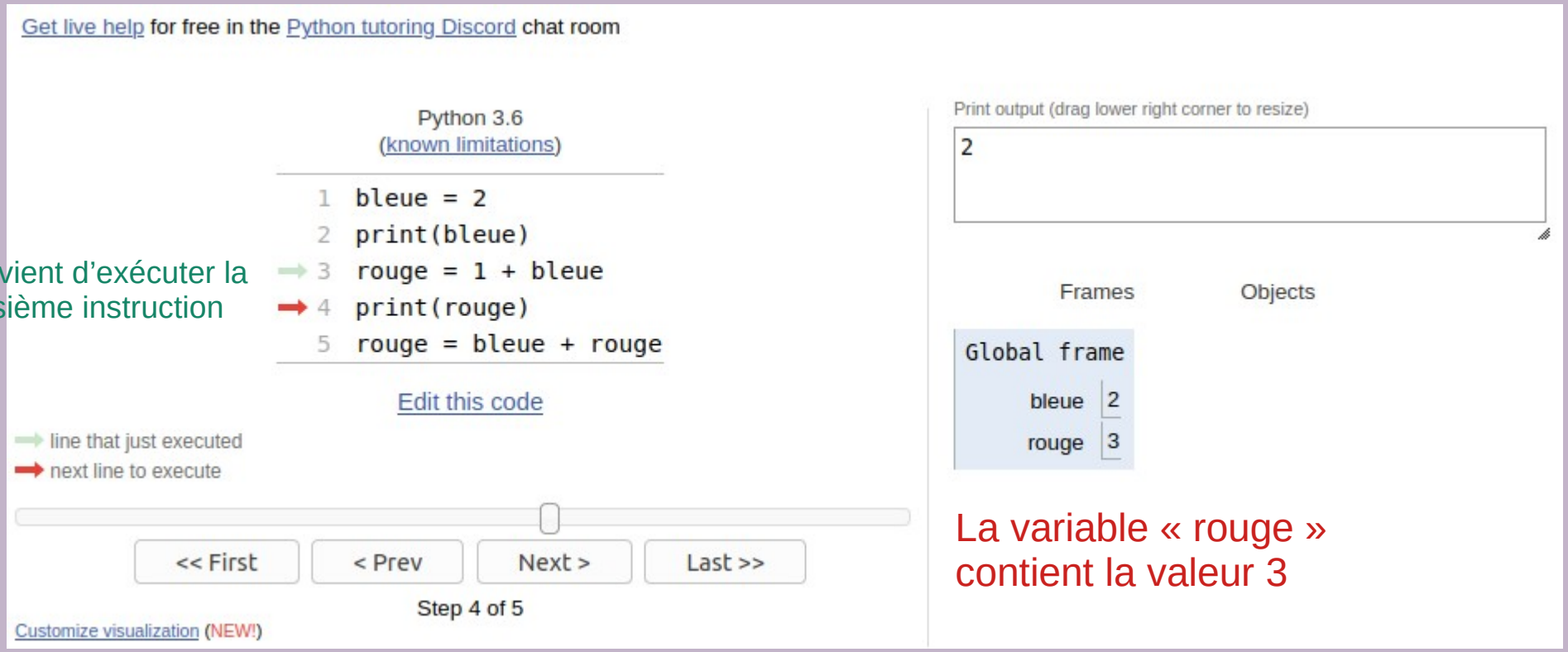


Exécution pas à pas (4)

On vient d'exécuter la troisième instruction



Exécution pas à pas (4)



On vient d'exécuter la troisième instruction

La variable « rouge » contient la valeur 3

Visualiser l’exécution en mode débranché

- Utiliser 3 élèves :
  - Un élève interprète les instructions du programme
  - Un élève gère la mémoire (sur une partie du tableau)
  - Un élève gère l’affichage (sur une autre partie du tableau)

Quelques conseils pédagogiques sur les variables et l'affectation

- N'introduire les variables qu'après avoir introduit la notion d'expression
- Multiplier les exercices simples, similaires
- Insister sur l'importance du nommage des variables
- Montrer comment l'introduction de variable peut :
  - aider à rendre lisible une expression compliquée

    Exemple :
    - plutôt que :

      ```
      r1 = (- b - √(b2-4ac))/2a
      ```
    - Préférer :

      ```
      delta = b2 - 4ac
      r1 = (-b - √(delta))/2a
      ```
  - éviter de faire 2 fois le même calcul

    ```
    r1 = (-b - √(delta))/2a
    r2 = (-b + √(delta))/2a
    ```

Et si on parlait un peu… typage ?

Retour sur les fonctions et opérateurs en mathématiques

- Un opérateur (souvent appelé loi de composition) n'est ni plus ni moins qu'une fonction avec une notation infixe
- En mathématiques, une fonction est définie sur un domaine, vers un codomaine
- Exemple :

$$+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$x , y \mapsto x + y$$

Retour sur les fonctions et opérateurs en mathématiques

- Un opérateur (souvent appelé loi de composition) n'est ni plus ni moins qu'une fonction avec une notation infixe
- En mathématiques, une fonction est définie sur un domaine, vers un codomaine
- Exemple :

$$+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$x \ , \ y \mapsto x + y$$

Typage de la fonction

Retour sur les fonctions et opérateurs en mathématiques

- Un opérateur (souvent appelé loi de composition) n'est ni plus ni moins qu'une fonction avec une notation infixe
- En mathématiques, une fonction est définie sur un domaine, vers un codomaine
- Exemple :

$$+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$x , y \mapsto x + y$$

Typage de la fonction

Les valeurs sont donc aussi typées

Typage en informatique

- Comme en mathématiques :
  - Les valeurs sont typées
  - Les fonctions sont typées
- Types de base courants :
  - Entiers (sous-ensemble fini de $\mathbb{N}$)
  - « Flottants » (sous-ensemble fini des nombres dyadiques)
  - Booléens
  - Chaînes de caractères
- Appliquer une fonction hors de son domaine de définition provoque une erreur
- Suivant les langages, les variables peuvent être typées (donc ne contenir que les données d'un type précis) ou non
- Des fonctions peuvent être définies sur plusieurs domaines, avec des comportements différents (notion de *surcharge*)

Typage : exemple en Python

```
>>> type(3)
<class 'int'>
>>> type('abc')
<class 'str'>
>>> type(3.14)
<class 'float'>
>>> 3 + 2
5
>>> 2 + 'ab'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 'ab' + 'cd'
'abcd'
```

Et concrètement, le typage ?

Et concrètement, le typage ?
C’est ça !

Penser informatiquement

La pensée informatique (1)

- Historique du terme *Computational thinking*
  - Terme introduit en 1980 par Seymour Papert
  - Popularisé en 2006 par Jeannette Wing
- Définition

  Le terme *Computational Thinking* désigne l'ensemble des techniques mises en œuvre pour arriver à faire résoudre des problèmes à des ordinateurs (techniques allant de la formalisation des problèmes à la description de méthodes de résolution).
- Intérêt

  Guider les enseignants en informatique dans la définition de leurs objectifs d'apprentissage et dans la vision de l'informatique qu'ils doivent faire passer : les compétences acquises dans les cours d'informatique seront utiles dans tous les domaines.

La pensée informatique (2)

Exemples de notions couvertes

- Reformuler un problème compliqué en un problème plus simple que l'on sait résoudre en utilisant des techniques de transformation, de réduction ou de simulation
- Penser récursivement
- Interpréter le code comme une donnée et inversement
- Évaluer le coût et la puissance de l'appel de procédure
- Juger de l'esthétique d'un programme
- Utiliser les invariants pour décrire le comportement d'un système
- Utiliser des heuristiques pour trouver une solution
- Planifier, apprendre et ordonnancer en présence d'incertain
- Savoir faire des compromis entre temps et espace

Les concepts ADAGE

- Quid ?

  Résumé de la notion de « pensée informatique » introduits en 2013 par Selby et Woolard, popularisé en 2017 par V. Dagiene

- Acronyme
  - **A**bstraction
  - **D**écomposition
  - Pensée **A**lgorithmique
  - **G**énéralisation
  - **É**valuation

A comme **A**bstraction

- Définition

  Être capable, dans la modélisation d'un problème, de ne garder que les données pertinentes pour la résolution du problème. Il devient ainsi plus facile de raisonner sur le problème et de le résoudre.

- Liens avec les mathématiques

  La même compétence est requise dans la formalisation nécessaire à la résolution d’un problème en mathématiques

D comme **D**écomposition

- Définition

  La décomposition est la compétence qui consiste à être capable de décomposer un problème complexe en sous-problèmes plus simples, et ce, récursivement, jusqu'à tomber sur des problèmes simples que l'on sait résoudre.

  Cette compétence est par exemple mise en œuvre dans les 2 cas suivants :

  - Démarche de conception descendante
  - Récursivité

- Liens avec les mathématiques

  La même compétence est nécessaire dans la résolution d’un problème de mathématiques nécessitant plusieurs étapes

A comme pensée **A**lgorithmique

- Définition

  La pensée algorithmique est une compétence qui consiste essentiellement à comprendre comment une succession d'étapes permet d'atteindre un objectif. Même si la définition de base ne parle que de l'aspect séquentiel, il semble logique d'ajouter à cette notion les concepts d'itération et de conditionnelle.

- Liens avec les mathématiques
  - Comprendre une démonstration, c’est appliquer un algorithme de preuve
  - Déterminer les extrema locaux d’une fonction, cela se fait en suivant un algorithme précis

A comme pensée **A**lgorithmique

Clés essentielles :

- Faire lire des programmes
- Dérouler à la main des programmes

A / T

Quelques vieux algorithmes en lien avec les mathématiques :

- Algorithme d'Euclide (IVè siècle avant notre ère)
- Approximation de π par des polygones à n côtés (IIIè siècle avant notre ère)
- κόσκινον Ἐρατοσθένους , IIIè siècle avant notre ère, mais première mention au IIè siècle de notre ère

G comme **G**énéralisation

- Définition

  La généralisation est une compétence qui consiste à savoir reconnaître que 2 problèmes différents ne sont en fait que des cas particuliers d'un même problème et ainsi, à savoir produire une solution pour tous les problèmes de la même famille.

- Liens avec les mathématiques
  - Comprendre qu’un théorème peut être appliqué de multiples fois, dans des situations différentes

E comme **É**valuation

- Définition

  L'évaluation est une compétence qui consiste à être capable de juger la solution produite : est-elle correcte ? est-elle suffisamment efficace en temps et et en espace ? Le logiciel est-il facile à utiliser ?

  Cette compétence est donc liée aux notions de complexité, de test, de qualité de code, de vérification. Mais avant tout, elle nécessite de savoir évaluer ce que produit un algorithme, en terme de valeur et en terme de type.

- Liens avec les mathématiques
  - Savoir juger si une démonstration est correcte
  - Savoir juger si les hypothèses pour appliquer un théorème sont vérifiées

En guise de conclusion ?

Quelques écueils classiques

- Le symbole « = » pour l'affectation utilisé dans la plupart des langages « modernes » est perturbant

  ex : `i = i + 1` ????

- Un nom de variable à gauche ou à droite du symbole d'affectation n'a pas le même sens
- Une affectation stocke la valeur de l'expression dans la variable, et non l'expression en elle-même

  ex : `a = 2 ; b = 2 * a ; print(b) ; a = 3 ; print(b)`

Ce n’est que le commencement !

Machine ?

Que contient réellement une variable ?

Processus ?

Langage ?

L’art poétique ?

Efficacité ?

Invariant ?

Leslie Lamport ?

Algorithme ?

Esthétique ?

Expression ?

Python ?

Classe ?

Récursivité ?

Démarche descendante ?

Variable typée ?