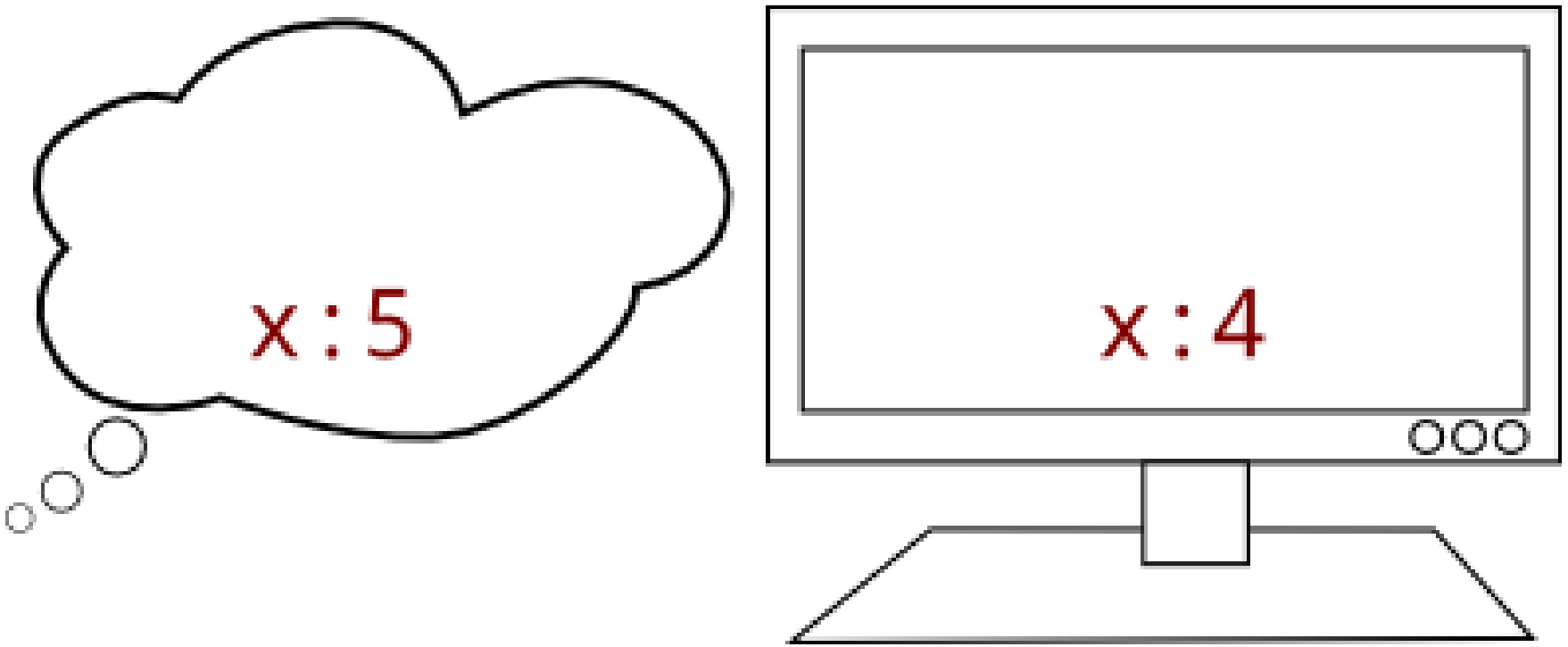
Quelques astuces de programmation

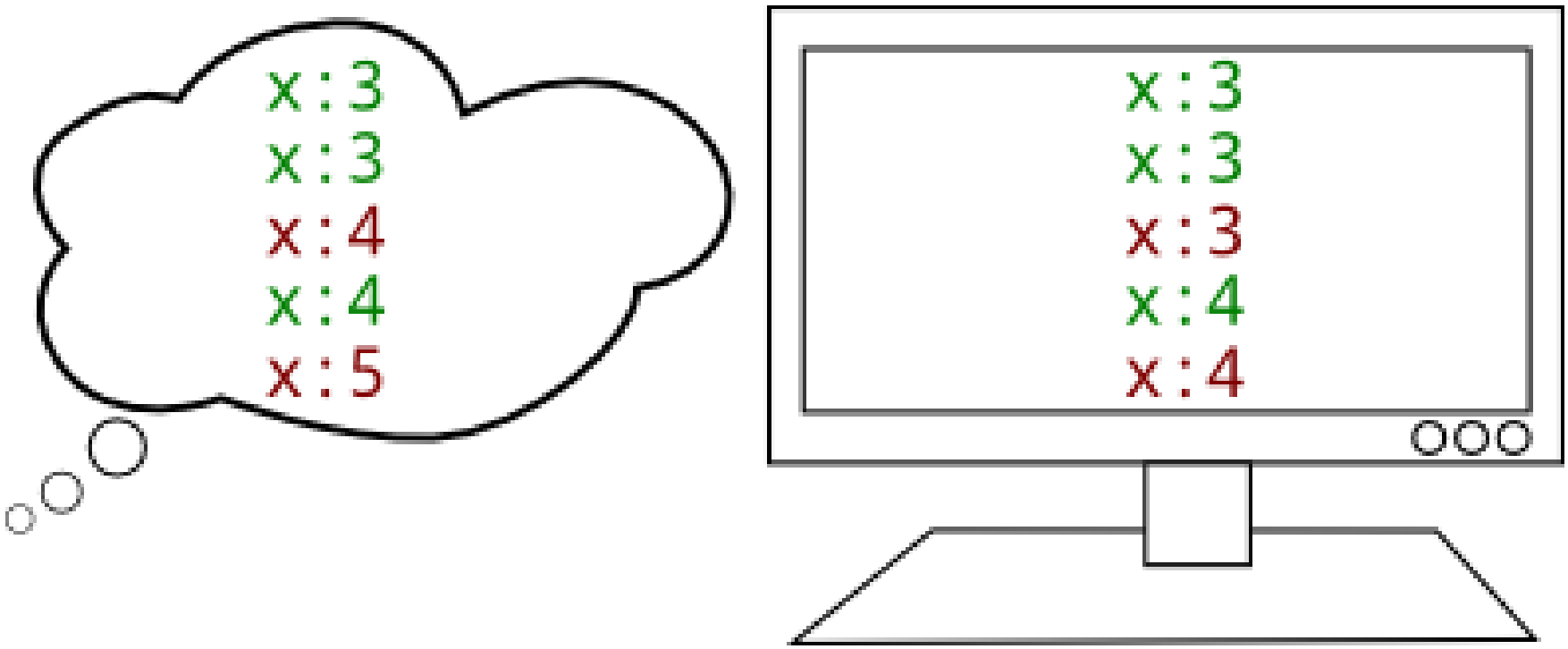
Zhentao Li

Comment trouver des erreurs



- L'ordinateur est un peu trop rapide et on pense connaître ce qu'il s'est passé (qu'il a exécuté les lignes dans l'ordre prévu, avec les effets attendus).
- Mais, en fait, quand une erreur se produit, c'est qu'il y une différence entre ce qu'on pense l'ordinateur a fait et ce qu'il s'est vraiment produit.

Comment trouver les erreurs



- Dans ce cas il faut trouver la différence. Pour ce faire, on peut exécuter le programme ligne par ligne.
- Le programme `pdb` peut aider à faire cela mais pour des programmes courts, utiliser l'interpréteur « à la main » suffit.

Exemple

```
n = 5
y = 0
z = 0
for i in range(n):
    z += 3*y + 3*i + 1
    y += 2*i + 1
```

```
>>> n = 5
>>> y = 0
>>> z = 0
>>> for i in range(n):
...     z += 3*y + 3*i + 1
...     y += 2*i + 1
...
>>> y
25
>>> z
125
```

On peut examiner le résultat final mais on ne sait pas ce qu'il s'est passé.

Exécuter un programme ligne par ligne: les boucles

```
n = 5
y = 0
z = 0
for i in range(n):
    z += 3*y + 3*i + 1
    y += 2*i + 1
```

- Si on fait du copier-coller d'une instruction "for", par exemple, toute la boucle va exécuter et on ne verra pas les étape intermédiaires.
- Dans ce cas, on peut « casser » la boucle en mettant la variable itérée à la bonne valeur et ensuite lancer ligne par ligne l'intérieur de la boucle.

Exécuter la boucle une ligne à la fois: les boucles

```
>>> n = 5
>>> y = 0
>>> z = 0
>>> range(n)
[0, 1, 2, 3, 4]
>>> i = 0
>>> z += 3*y + 3*i + 1
>>> z
1
>>> y += 2*i + 1
>>> y
1
>>> i = 1
>>> z += 3*y + 3*i + 1
>>> z
8
>>> y += 2*i + 1
>>> y
4
>>> i = 2
```

```
>>> z += 3*y + 3*i + 1
>>> z
27
>>> y += 2*i + 1
>>> y
9
>>> i = 3
>>> z += 3*y + 3*i + 1
>>> z
64
>>> y += 2*i + 1
>>> y
16
>>> i = 4
>>> z += 3*y + 3*i + 1
>>> z
125
>>> y += 2*i + 1
>>> y
25
```

Exécuter un programme ligne par ligne: condition

- On évalue la condition lorsque on y arrive pour décider si les prochaines lignes devraient être exécutés.

```
if x > 2 and y < 3:
    x += 1
```

```
>>> x > 2 and y < 3
False
```

Exécuter un programme ligne par ligne: les fonctions

- Pour les appels aux fonctions,

```
def f(a, b, c, d):
    somme = a + b + c + d
    return somme

v = f(w, x, y, z)
```

```
>>> # ligne def f(a, b, c, d):
>>> a = w
>>> b = x
>>> c = y
>>> d = z
>>> # ligne somme = a + b + c + d
>>> somme = a + b + c + d
>>> # ligne return somme
>>> v = somme
```

Exécuter un programme ligne par ligne: lignes complexes (avancée)

- Pour une ligne complexe, on peut évaluer chaque composante individuellement. Par exemple, évaluer chaque paramètre d'une fonction avant de l'exécuter.

```
v = f(g(w), h(x+y))
```

```
>>> sortie_g = g(w)
>>> entree_h = x+y
>>> sortie_h = h(entree_h)
>>> sortie_f = f(sortie_g, sortie_h)
>>> v = sortie_f
```

Création d'un programme

Plusieurs possibilités ou styles différents.

- Écrire le tout d'un coup et le lancer après.
  - Avantage: Bonne vue d'ensemble. Nous donne l'idée en tête même si le programme ne tourne pas.
  - Désavantage: Difficile à déboguer une fois qu'on lance le programme. Les erreurs peuvent provenir de n'importe quel segments et de plusieurs segments en même temps.
- Écrire une fonction à la fois et tester à chaque fois.
  - Cette méthode marche mieux si on crée les tests et les résultats attendus avant d'écrire la fonction (mais prends plus de temps).
- Écrire tout dans l'interpréteur et recopier dans un module les lignes qui marchent.
  - Avantage: Réponse immédiate. Bon quand on utilise un module inconnu.
  - Désavantage: L'idée globale peut être perdue, interrompue.
- Autre