Bases de la programmation impérative (BPI)

CM2 : boucles, complexité, types abstraits et structures de données

Manuel Selva

À la recherche de quelques étudiants pour ...

- Former des migrants à l'utilisation de l'informatique
- https://coformer.univ-grenoble-alpes.fr/
- ECTS SHEME
- Octobre/Décembre et Février/Avril

Quel est votre ressenti ?

- Comment trouvez vous le rythme des TD ?
  - trop lent, bon, trop rapide

Quel est votre ressenti ?

- Comment trouvez vous le rythme des TD ?
  - trop lent, bon, trop rapide
- Comment trouvez vous le rythme des TP ?
  - trop lent, bon, trop rapide

Quel est votre ressenti ?

- Comment trouvez vous le rythme des TD ?
  - trop lent, bon, trop rapide
- Comment trouvez vous le rythme des TP ?
  - trop lent, bon, trop rapide
- Qui n'a pas d'environnement Linux sur sa machine personnelle ?

Quel est votre ressenti ?

- Comment trouvez vous le rythme des TD ?
  - trop lent, bon, trop rapide
- Comment trouvez vous le rythme des TP ?
  - trop lent, bon, trop rapide
- Qui n'a pas d'environnement Linux sur sa machine personnelle ?

Sommaire du jour

Séquences et boucles

Combien ça coûte ?
- Premier exemple
- Problème
- Explications

Structures de données et types abstraits

À retenir

Jusqu'à présent

- Des scalaires, des chaînes de caractères de petite taille et des tuples de petite taille
- Pas de conteneur de taille variable
- Pas de `list` (sans **E**, donc c'est le type Python)
- Pas de boucle

Qu'est-ce qu'une séquence Python ?

"These represent finite ordered sets indexed by non-negative numbers.

The built-in function `len()` returns the number of items of a sequence.

When the length of a sequence is n, the index set contains the numbers $0, 1, \ldots, n-1$.

Item i of sequence a is selected by `a[i]`."[1]

1. https://docs.python.org/3/reference/datamodel.html#the-standard-type-hierarchy

Quelles séquences Python connaissez vous ?

? ? ? ?

Les séquences Python

- chaînes de caractères : `s = "BPI"; s[2]`

Les séquences Python

- chaînes de caractères : `s = "BPI"; s[2]`
- tuples : `t = ("C", 1, "peu"); t[1]`

Les séquences Python

- chaînes de caractères : `s = "BPI"; s[2]`
- tuples : `t = ("C", 1, "peu"); t[1]`
- `list` : `l = ["intéressant", "non?"]; l[0]`

Les séquences Python

- chaînes de caractères : `s = "BPI"; s[2]`
- tuples : `t = ("C", 1, "peu"); t[1]`
- `list` : `l = ["intéressant", "non?"]; l[0]`
- et bien d'autres encore (affaire à suivre)

Comment parcourir une séquence ?

À l'ancienne : ok dans tous les langages impératifs

```
i = 0
while i < len(seq):
    elem = seq[i]
    print(elem)
    i += 1
```

Comment parcourir une séquence ?

À l'ancienne : ok dans tous les langages impératifs

```
i = 0
while i < len(seq):
    elem = seq[i]
    print(elem)
    i += 1
```

```
for (i = 0, i < len(seq), i=i+1):
    elem = seq[i]
    print(elem)
```

Comment parcourir une séquence ?

À l'ancienne : ok dans tous les langages impératifs

```
i = 0
while i < len(seq):
    elem = seq[i]
    print(elem)
    i += 1
```

```
for (i = 0, i < len(seq), i=i+1):
    elem = seq[i]
    print(elem)
```

(sauf en Python)

Comment parcourir une séquence ?

À l'ancienne : ok dans tous les langages impératifs

```
i = 0
while i < len(seq):
    elem = seq[i]
    print(elem)
    i += 1
```

```
for (i = 0, i < len(seq), i=i+1):
    elem = seq[i]
    print(elem)
```

(sauf en Python)

En Python

```
for i in range(len(seq)):
    elem = seq[i]
    print(elem)
```

Comment parcourir une séquence ?

À l'ancienne : ok dans tous les langages impératifs

```
i = 0
while i < len(seq):
    elem = seq[i]
    print(elem)
    i += 1
```

```
for (i = 0, i < len(seq), i=i+1):
    elem = seq[i]
    print(elem)
```

(sauf en Python)

En Python

```
for i in range(len(seq)):
    elem = seq[i]
    print(elem)
```

```
for elem in seq:
    print(elem)
```

Sommaire du jour

Qui dit boucle ...

... dit programme éventuellement **"complexe"**

Qui dit boucle ...

... dit programme éventuellement **"complexe"**

Combien d'instructions machines pour ce programme sans boucle ?

```
a = 17
b = a * 42
... # avec 3767 lignes sans boucle
... # ni appel de fonction
```

Qui dit boucle ...

... dit programme éventuellement **"complexe"**

Combien d'instructions machines pour ce programme sans boucle ?

```
a = 17
b = a * 42
... # avec 3767 lignes sans boucle
... # ni appel de fonction
```

Combien d'instructions machines pour ce programme avec boucle ?

```
my_sum = 0
for i in range(len(seq)):
    elem = seq[i]
    my_sum += elem
```

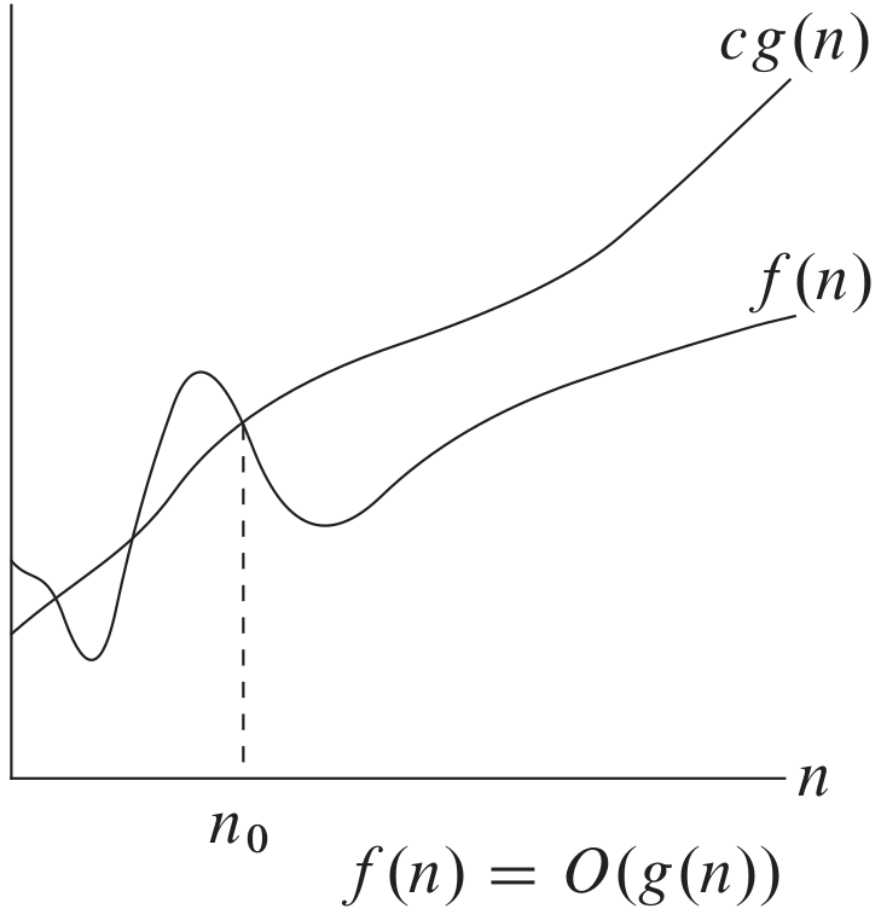
Qui connaît la notation O ?

Qui connaît la notation O ?

$$O(g(n)) = \{f(n) : \exists c, n0 >= 0 \mid 0 <= f(n) <= c * g(n) \forall n >= n0\}$$

Qui connaît la notation O ?

$$O(g(n)) = \{f(n) : \exists c, n0 >= 0 \mid 0 <= f(n) <= c * g(n) \forall n >= n0\}$$

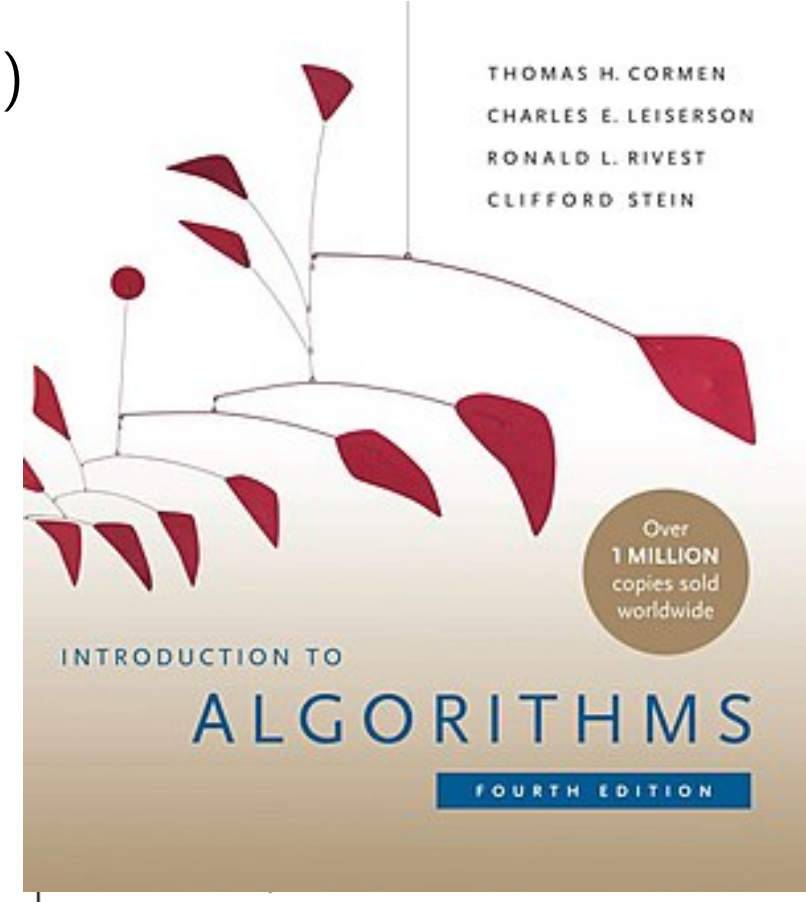


Qui connaît la notation O ?

$O(g(n)) = \{f(n)$ $g(n) \forall n >= n0\}$

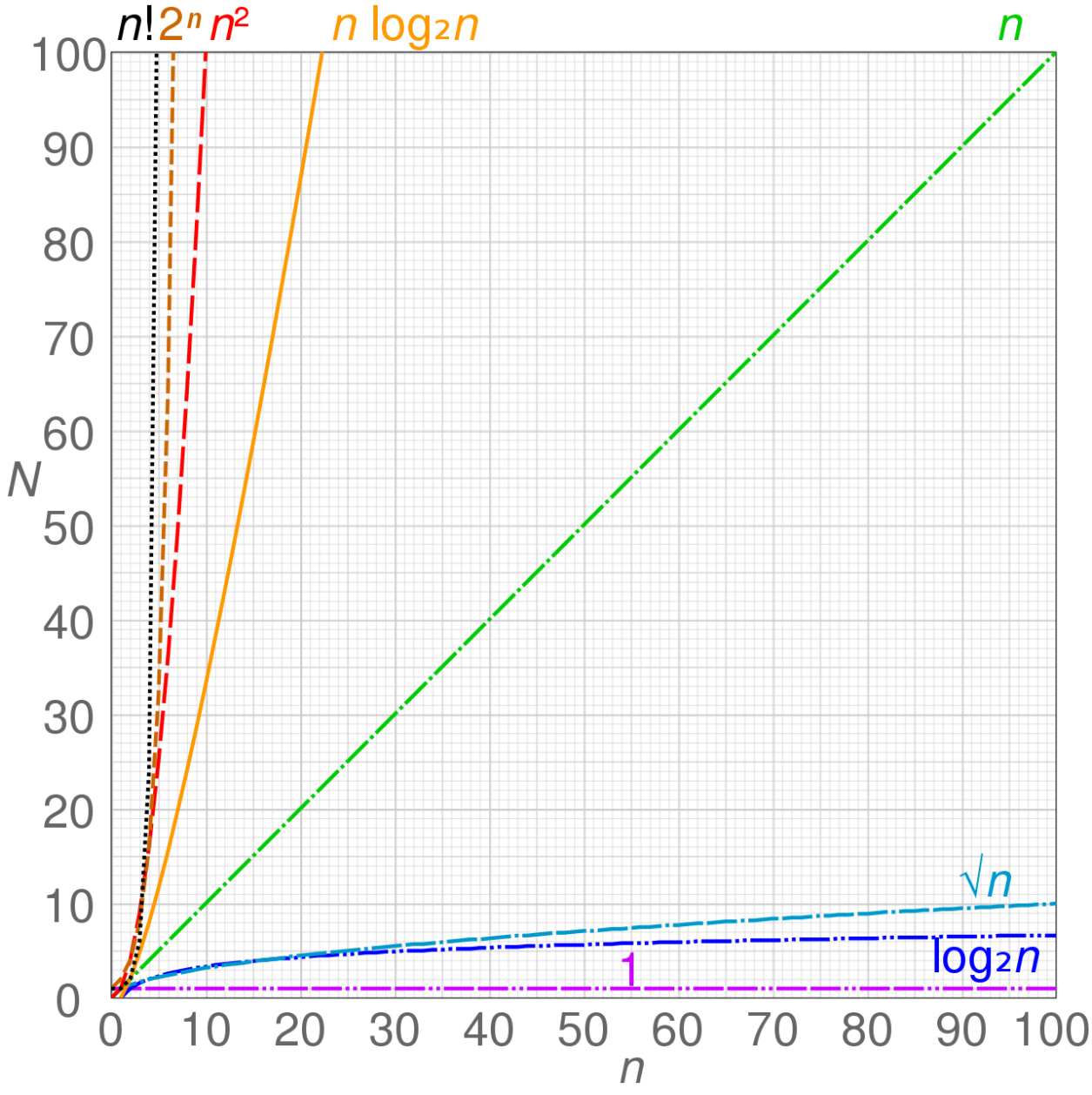
(n)

(n)



"Having a solid base of algorithmic knowledge and technique is one characteristic that separates the truly skilled programmers from the novices."

Qui c'est qui gagne, et de combien ? (source Wikipedia)



Organisation pour la suite du cours

- 5mn pour résoudre **individuellement**
- 2mn pour échanger avec **un seul** voisin
- vote
- Xmn pour débattre
- vote
- Xmn pour débattre à nouveau
- Xmn pour comprendre la bonne réponse

Combien d'instructions machines pour do_complex_stuff ?

```
1  #!/usr/bin/env python3
2  from random import randint
3  from random import seed
4
5  def is_odd(integr):
6      return bool(integr % 2)
7
8  def do_complex_stuff(nb):
9      r = []
10     m = 0
11     for _ in range(nb):
12         i = randint(0, nb)
13         if is_odd(i):
14             r.insert(m, i)
15             m += 1
16         else:
17             w = len(r)
18             r.insert(w, i)
19     return r
```

```
20
21 seed(42)
22 n = input("Enter a number:\n")
23 l = do_complex_stuff(int(n))
24 f = open("out.data", "w")
25 for elem in l:
26     print(elem, file=f)
27 f.close()
```

A. $100 * nb$

B. $100 * (nb * x)$ avec $x < constante$

C. $100 * (nb * x)$ avec x dépendant des résultats de `randint`

D. plus que les 3 réponses ci-dessus

E. autre

Par curiosité ...

... essayons le programme avec `nb = 1.000.000`

Analysons le programme

```
1  #!/usr/bin/env python3
2  from random import randint
3  from random import seed
4
5  def is_odd(integr):
6      return bool(integr % 2)
7
8  def do_complex_stuff(nb):
9      r = []
10     m = 0
11     for _ in range(nb):
12         i = randint(0, nb)
13         if is_odd(i):
14             r.insert(m, i)
15             m += 1
16         else:
17             w = len(r)
18             r.insert(w, i)
19     return r
```

```
20
21 seed(42)
22 n = input("Enter a number:\n")
23 l = do_complex_stuff(int(n))
24 f = open("out.data", "w")
25 for elem in l:
26     print(elem, file=f)
27 f.close()
28
29
30
31
32
33
34
35
36
37
38
```

Que peut on faire avec une `list` Python ?

Que peut on faire avec une `list` Python ?

- créer : `etus = ["Zoé", "Noé", "Alexia", "Maxime"]`

Que peut on faire avec une `list` Python ?

- créer : `etus = ["Zoé", "Noé", "Alexia", "Maxime"]`
- accéder à un élément : `etus[3]`

Que peut on faire avec une `list` Python ?

- créer : `etus = ["Zoé", "Noé", "Alexia", "Maxime"]`
- accéder à un élément : `etus[3]`
- modifier un élément : `etus[2] = "Mehdi"`

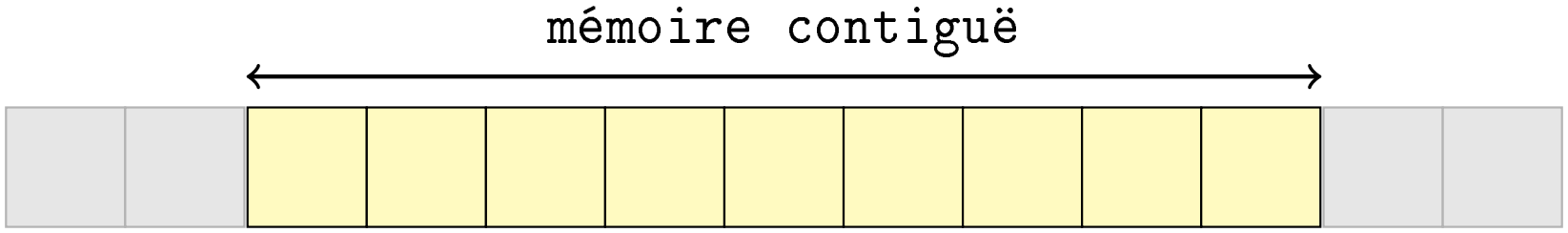
Que peut on faire avec une `list` Python ?

- créer : `etus = ["Zoé", "Noé", "Alexia", "Maxime"]`
- accéder à un élément : `etus[3]`
- modifier un élément : `etus[2] = "Mehdi"`
- la faire grandir :
  - `etus.append("Manu")`
  - `etus.insert(1, "Céline")`

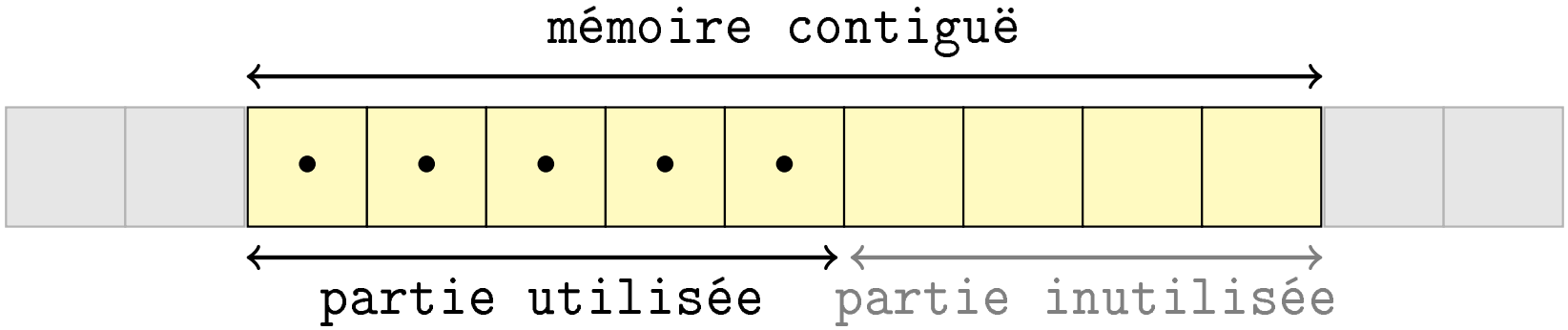
Que peut on faire avec une `list` Python ?

- créer : `etus = ["Zoé", "Noé", "Alexia", "Maxime"]`
- accéder à un élément : `etus[3]`
- modifier un élément : `etus[2] = "Mehdi"`
- la faire grandir :
  - `etus.append("Manu")`
  - `etus.insert(1, "Céline")`
- la faire rapetisser : `etus.remove("Manu")`

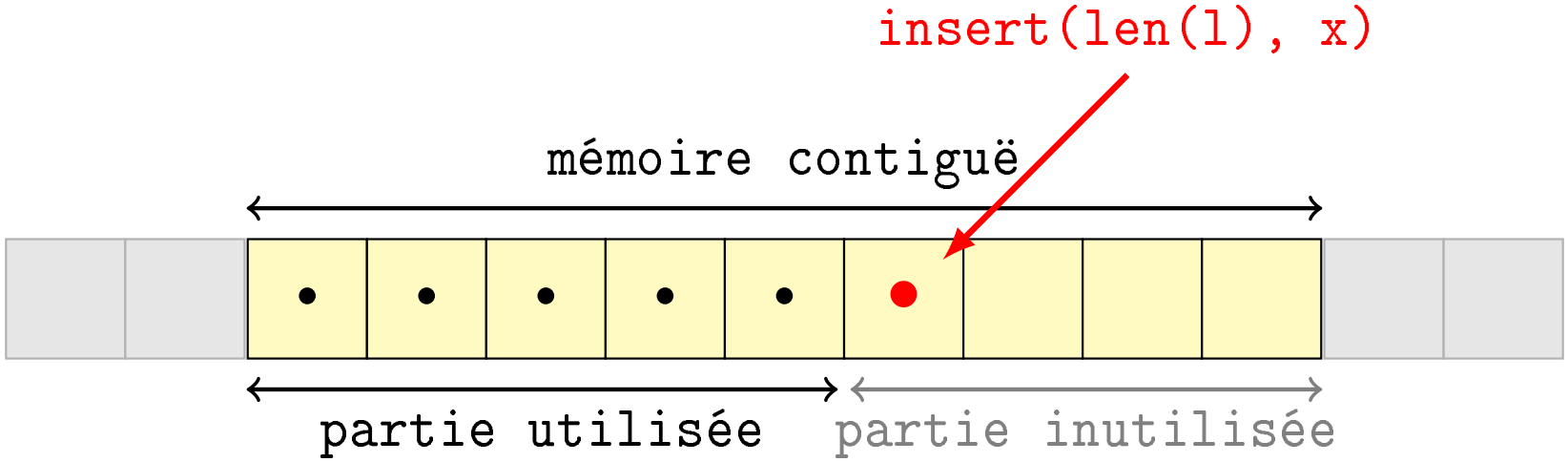
Comment est implémentée une `list` Python ?



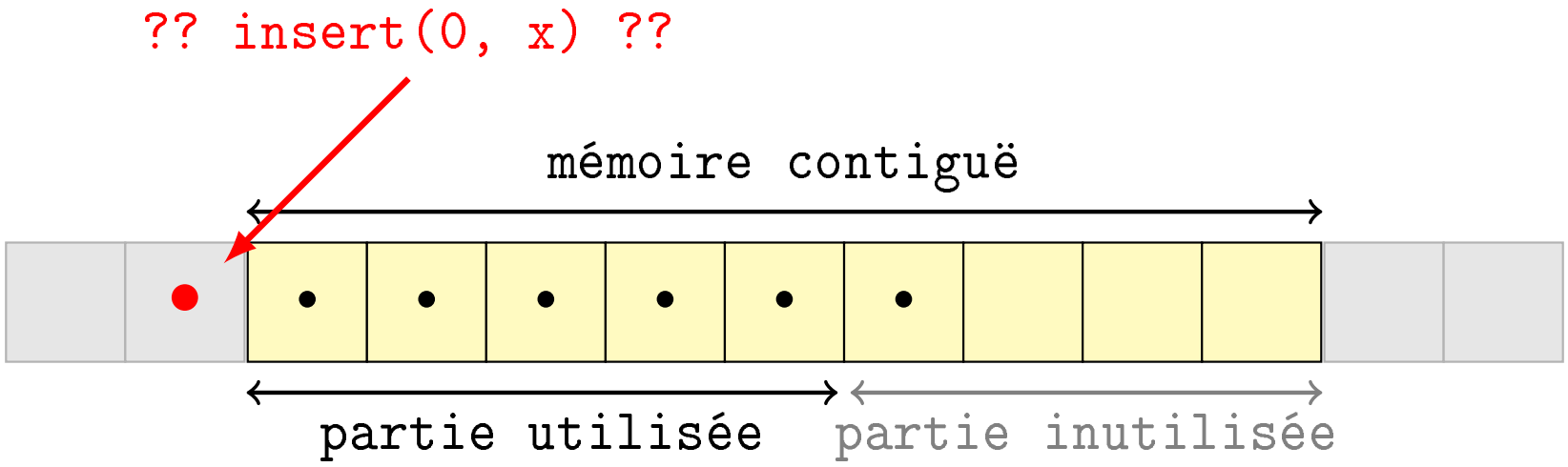
Comment est implémentée une `list` Python ?



Comment est implémentée une `list` Python ?

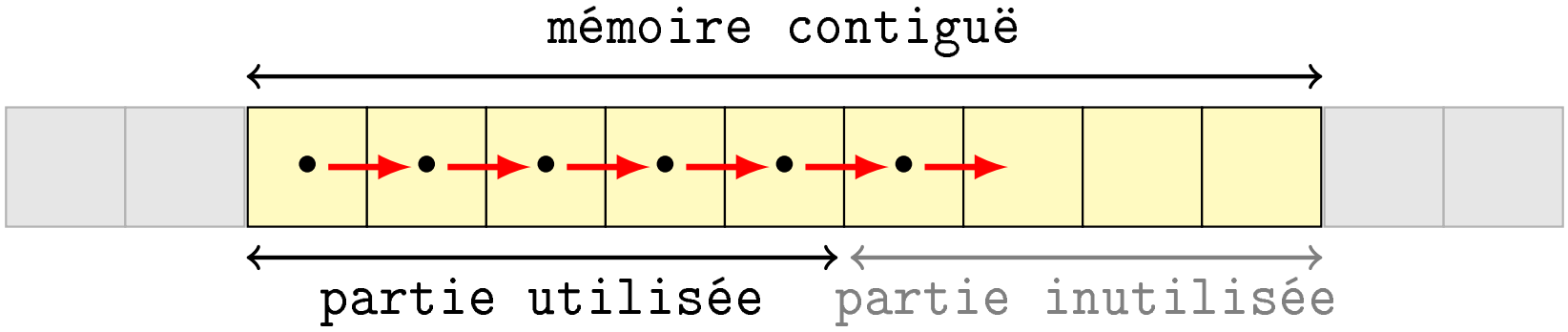


Comment est implémentée une `list` Python ?



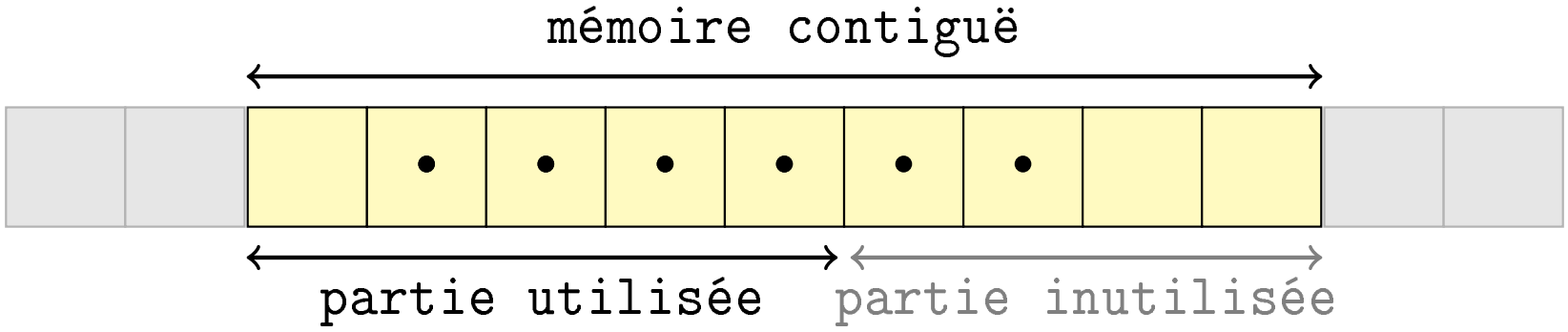
Comment est implémentée une `list` Python ?

`insert(0, x)`

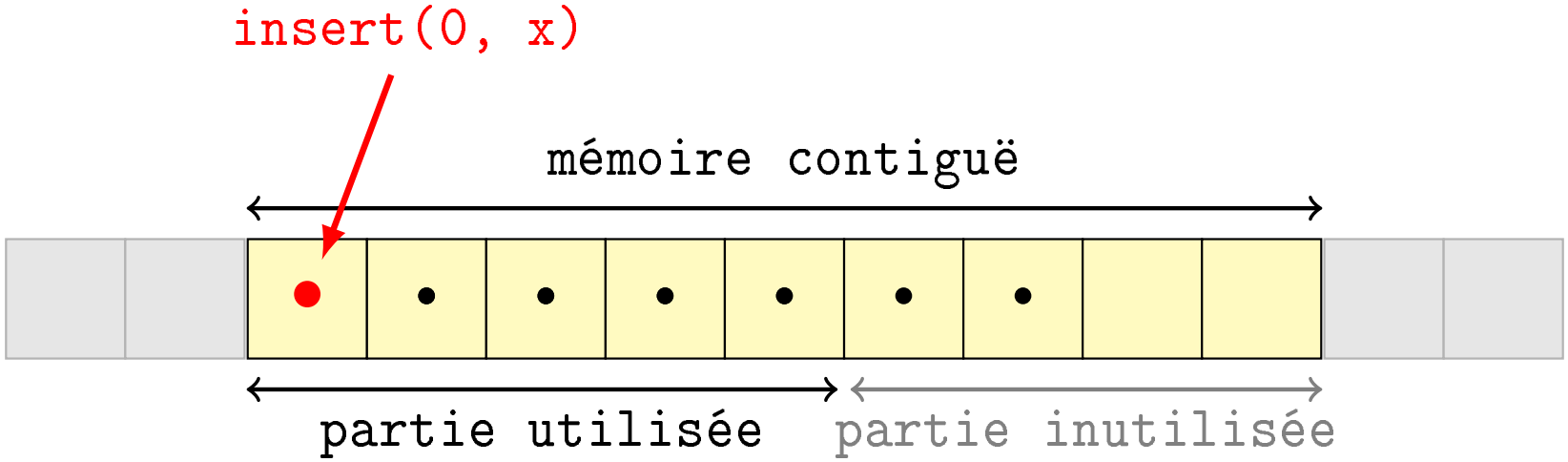


Comment est implémentée une `list` Python ?

`insert(0, x)`



Comment est implémentée une `list` Python ?



La bonne réponse

Les propositions

A. $100 * nb$

B. $100 * (nb * x)$ avec $x < constante$

C. $100 * (nb * x)$ avec x dépendant des résultats de `randint`

D. **plus que les 3 réponses ci-dessus**

E. autre

Comment améliorer les choses ?

```
1  #!/usr/bin/env python3
2  from random import randint
3  from random import seed
4
5  def is_odd(integr):
6      return bool(integr % 2)
7
8  def do_complex_stuff(nb):
9      odd = []
10     ev = []
11     for _ in range(nb):
12         i = randint(0, nb)
13         if is_odd(i):
14             odd.append(i)
15         else:
16             ev.append(i)
17     r = list(odd)
18     r.extend(ev)
19     return r
20
```

```
21 seed(42)
22 n = input("Enter a number:\n")
23 l = do_complex_stuff(int(n))
24 f = open("out-fast.data", "w")
25 for elem in l:
26     print(elem, file=f)
27 f.close()
28
29
30
31
32
33
34
35
36
37
38
39
```

Par curiosité ...

... essayons le nouveau programme avec `nb = 1.000.000`

et assurons nous que les deux programmes font bien la même chose en comparant les fichiers de sortie :

```
> diff out.data out-fast.data
```

Sommaire du jour

SDD et TA : définitions

Type abstrait (TA)

Spécification d'un type de données : ensemble des propriétés qui le définissent et des opérations qu'on peut effectuer sur lui.

SDD et TA : définitions

Type abstrait (TA)

Spécification d'un type de données : ensemble des propriétés qui le définissent et des opérations qu'on peut effectuer sur lui.

Structure de données (SDD)

Mise en œuvre concrète d'un TA.
La notion de coût d'une opération est associée à une SDD.

Revenons sur la `list` Python

`list` est une SDD

- car c'est une mise en œuvre concrète
- aussi nommée **tableau dynamique** ou **vecteur**

Revenons sur la `list` Python

`list` est une SDD

- car c'est une mise en œuvre concrète
- aussi nommée **tableau dynamique** ou **vecteur**

qui met en œuvre le TA

- séquence finie de taille variable
  - accès par index entier
  - modification de l'élément à un index donné
  - ajouts au début, au milieu et en fin
  - suppression au début, au milieu et en fin

Attention à la terminologie

Il n'y a pas de terminologie universelle et souvent TA et SDD sont confondus (comme en Python).

Attention à la terminologie

Il n'y a pas de terminologie universelle et souvent TA et SDD sont confondus (comme en Python).

Il faut donc toujours s'interroger pour savoir **de quoi parle-t-on ?**

Attention à la terminologie

Il n'y a pas de terminologie universelle et souvent TA et SDD sont confondus (comme en Python).

Il faut donc toujours s'interroger pour savoir **de quoi parle-t-on ?**

En BPI on se référera à :

https://bpi-etu.pages.ensimag.fr/2-iterations/adt_sdd.pdf

Sommaire du jour

À retenir

- Avec des boucles on peut écrire des programmes complexes
- ⚠ Ne pas faire n'importe quoi avec des `list` Python ⚠
- On va apprendre à maîtriser les TA et les SDD élémentaires
- On va **toujours s'interroger** sur le coût des fonctions que nous **implémenterons** et donc aussi des fonctions que nous **utiliserons** à partir de tout de suite !