Informatique 1
Introduction à la programmation

Mission 5 : introduction

Kim Mens Siegfried Nijssen Charles Pecheur

Immuable vs Muable

Chaînes de caractères (Strings)

Chaînes ne sont **pas** muables

```
s = ""
s.append ( "a" )
s.append ( "b" )
print ( s )
```

```
'str' object has no
  attribute 'append'
```

Listes

Listes sont muables

```
l = []
l.append ( 1 )
l.append ( 2 )
print ( l )
```

```
[ 1, 2 ]
```

Structures de Données Immuables

Pourquoi les structures de données immuables existent?

- Calculs sur les structures de données immuables sont souvent plus rapides
- Les structures de données immuables prennent moins de mémoire

Tuples

Tuple = liste immuable

Tuples

Tuples ne sont **pas** muables

```
t = ( 1, 2 )
```

Listes

Listes sont muables

```
l = [ 1, 2 ]
```

Tuples

Tuple = liste immuable

Tuples

Tuples ne sont **pas** muables

```
t = ( 1, 2 )
t[0] = 3
print ( t )
```

```
'tuple' object does
 not support item
 assignment
```

Listes

Listes sont muables

```
l = [ 1, 2 ]
l[0] = 3
print ( l )
```

```
[ 3, 2 ]
```

Tuples & Listes Imbriquées

```
>>> l = [ (0,1), (2,3), ([4,5],[6,7]) ]
>>> print ( l[0][0], l[1][0], l[2][0], l[2]
  [0][1] )

0 2 [4,5] 5
```

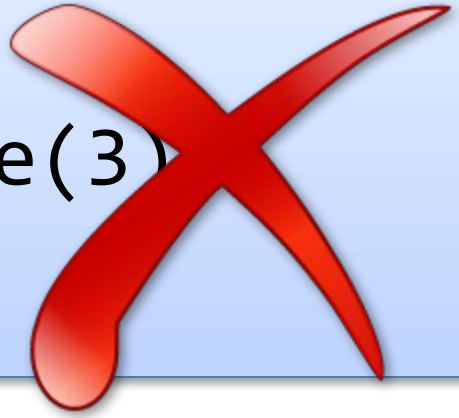
Création de Tuples

- On veut créer

  (0,1,2)

- Est-ce que ce code est correct?

```
t = ()
for j in range(3):
  t.append(j)
```



Création de Tuples

- On veut créer

  (0,1,2)

- Approche correcte

```
t = []
for j in range(3):
  t.append(j)
t =  tuple(t)
```

Conversion de type

Qu'est-ce que c'est, Code de Haute Qualité ?

- Code qui fait ce qui est attendu
- Code qui est efficace
- Code qui est lisible
- Code qui est facile à modifier
- Code qui est facile à étendre
- Code qui est facile à utiliser

par vous et par d'autres programmeurs

Comment d'écrire de Code de Haute Qualité ?

- Question très difficile; pas une seule solution
- On a déjà vu quelques recommandations
  - Pas copier – coller
  - Diviser le code en fonctions
  - Variables avec des noms lisibles
  - Écrivez des commentaires dans le code, en particulier une spécification

Docstring

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres dans
            la liste
  """
```

= Spécification de la fonction

La plupart du temps, on utilise pre/post-conditions dans ce cours

Docstring: Style Google

```
def multiplier ( l ):
  """ Retourne la multiplication des nombres dans la liste l

  Par exemple, multiplier ( [ 1, 2, 3 ] ) == 6

  Args:
      l: une liste de nombres
  Returns:
      un float qui représente la multiplication des nombres
      dans la liste l
  """
```

Utilisé dans beaucoup de projets

Plus long, plus détaillé

Docstring: Style Google

```
def fonction( paramètres ):
  """ Une ligne

  Déscription plus longue

  Args:
      nom de paramètre: déscription, souvent avec type
  Returns:
      Déscription de ce qui retourne la fonction
  """
```

Tests

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres
  """
  a = l[0]
  for v in l:
     a = a * v
  return a
```

Est-ce que ce programme est correct?

Idée: on va écrire du code pour vérifier si la fonction est correcte

Écrire des Tests

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres
  """
```

Étape 1: oubliez le code

Étape 2: identifiez des **cas de test** basé sur la spécification

Entrée (l)	Sortie (return)
[1,3,4]	12
[4,3,1]	12

Écrire des Tests

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres
  """
```

Étape 1: oubliez le code

Étape 2: identifiez des **cas de test** basé sur la spécification

Étape 3: écrivez de code pour exécuter le programme sur les cas de test

Programmer des Tests

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres
  """
  a = l[0]
  for v in l:
     a = a * v
  return a
```

Fonction pour tester multiplier

```
def test_multiplier ():
  assert multiplier ( [1, 3, 4] ) == 12, "Test 1"
  assert multiplier ( [3, 4, 1] ) == 12, "Test 2"
  assert multiplier ( [3, 4, 5] ) == 60, "Test 3"
  assert multiplier ( [3, 2, 2, 5] ) == 60, "Test 4"
```

Instruction qui évalue un test

Tests

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres
  """
  a = l[0]
  for v in l:
     a = a * v
  return a
```

- `multiplier ( [4, 3, 1] ) == 12?`
- Donne 48 -> le code n'est pas correct

Tests

```
def multiplier ( l ):
  """ pre: l une liste de nombres
      post: retourne la multiplication des nombres
  """
  a = l[0]
  for v in l:
     a = a * v
  return a
```

La qualité de tests est importante. Le test suivant ne montre pas d'erreur!

- `multiplier ( [1, 3, 4] ) == 12`

On doit trouver des cas "difficiles" pour lesquels il est probable qu'une fonction commettra une erreur

Assert

Notation:

`assert` *b*, *c*

Où

- *b* est une expression booléenne
- *c* est une chaîne de caractères

Algorithmes de Recherche

*"Un **algorithme** est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre une classe de problèmes"*

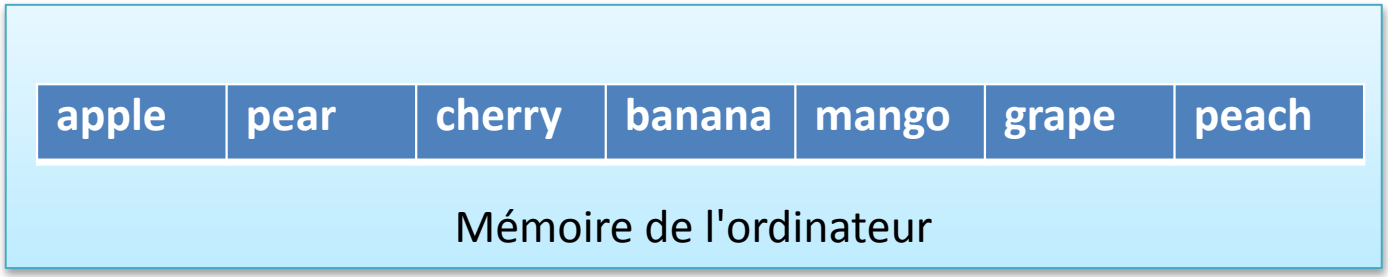
Un **problème de recherche** dans une liste est un problème de trouver un objet *x* bien défini dans la liste.

Par exemple, étant donnée

apple	pear	cherry	banana	mango	grape	peach

comment un peut déterminer si l'object *mango* se trouve dans la liste?

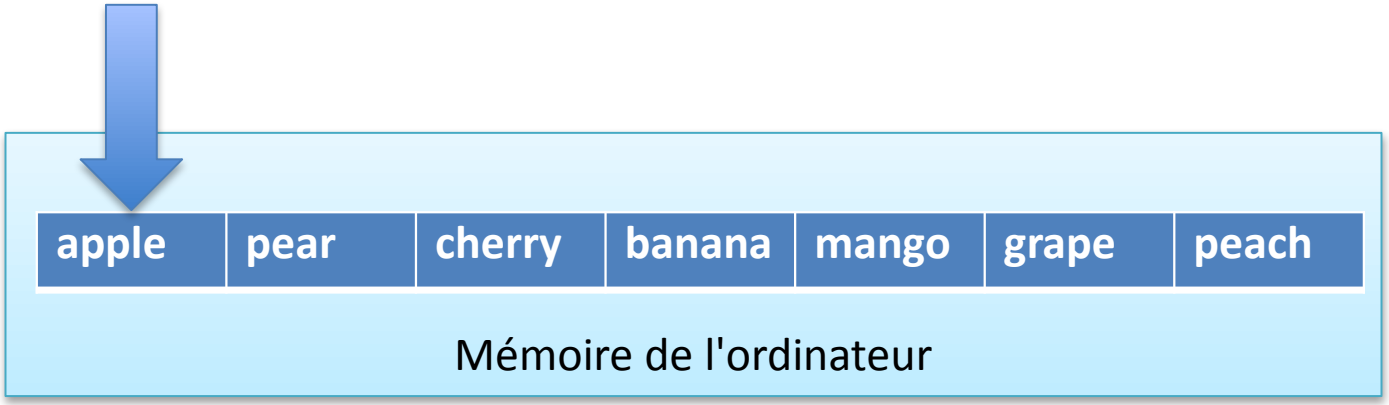
Recherche Séquentielle (Linéaire)




```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

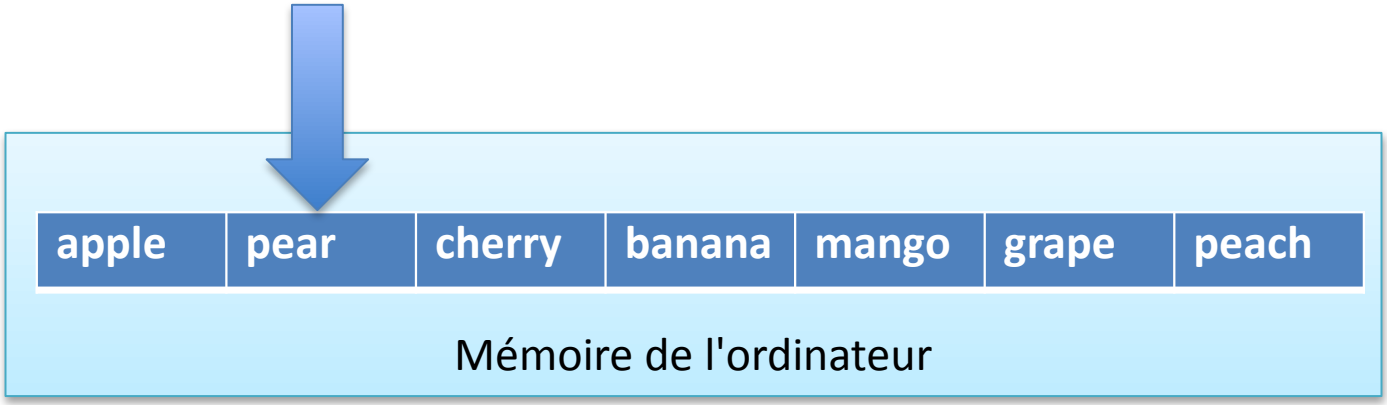
(1) Séquence (éléments)
(2) Condition de traitement
(3) Corps de la boucle

Recherche Séquentielle (Linéaire)



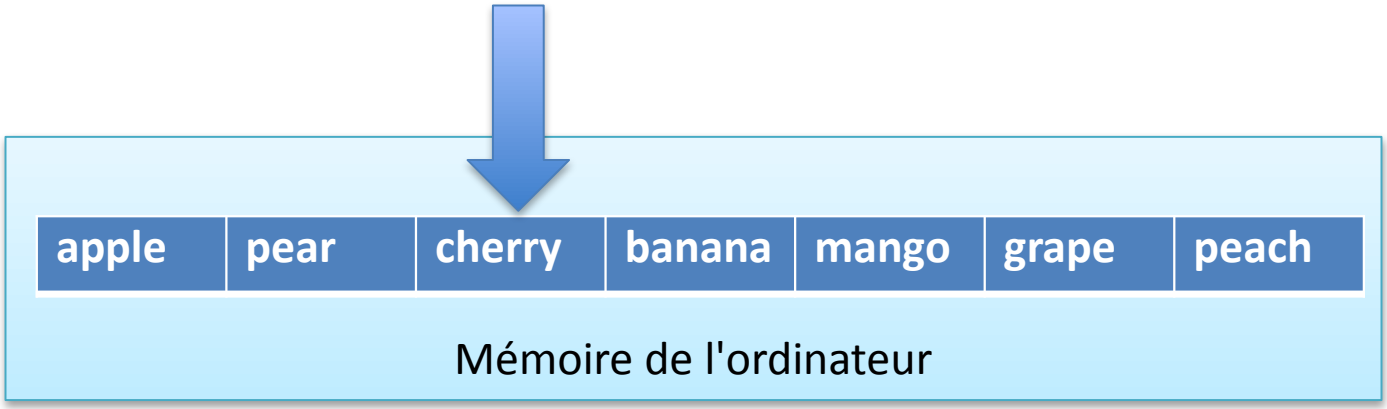

```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

Recherche Séquentielle (Linéaire)



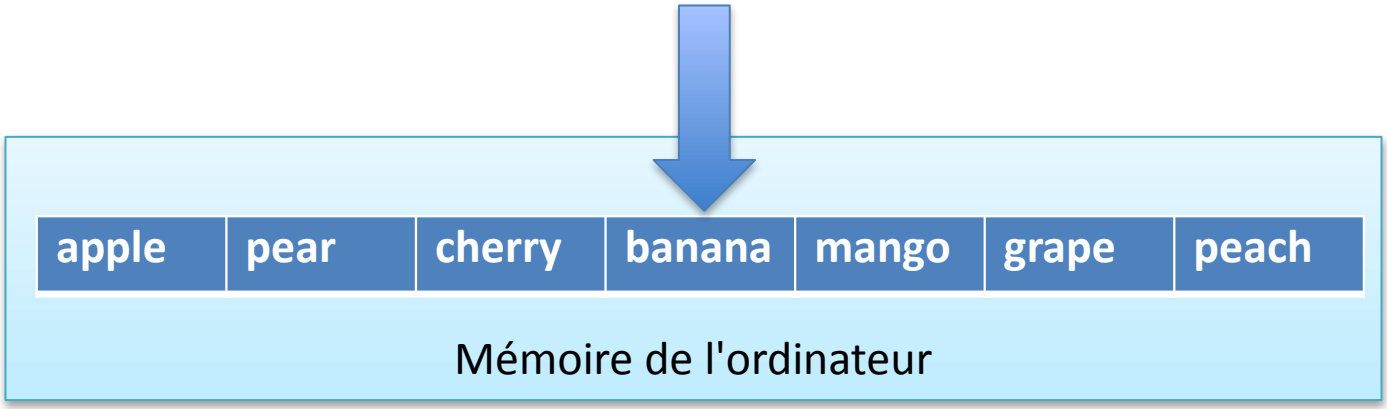

```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

Recherche Séquentielle (Linéaire)



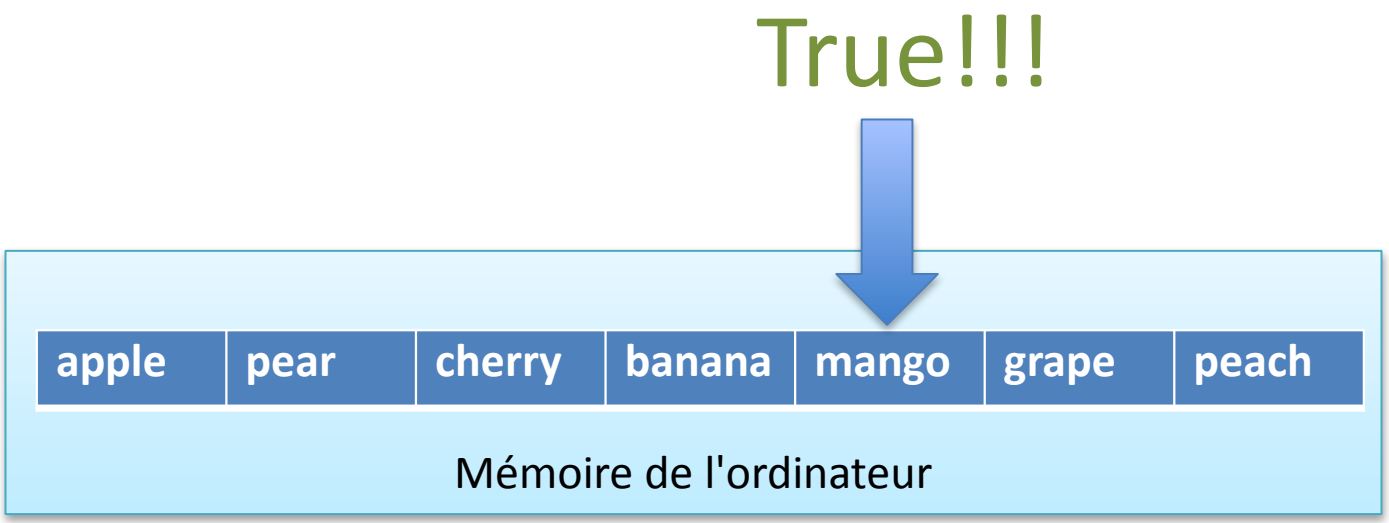

```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

Recherche Séquentielle (Linéaire)



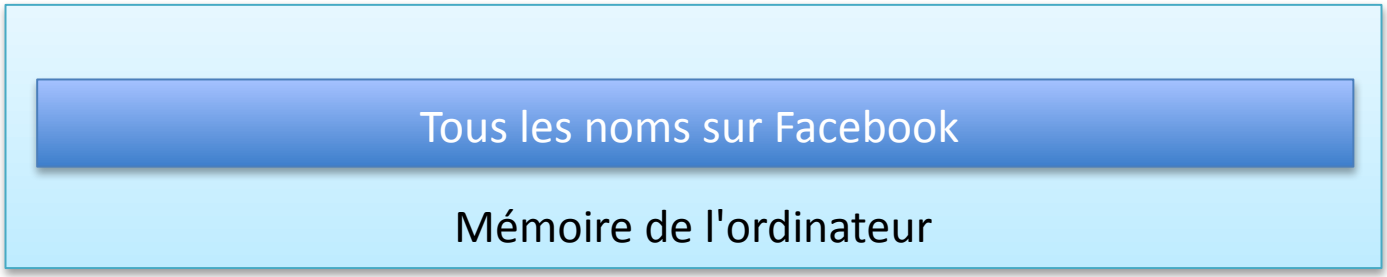

```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

Recherche Séquentielle (Linéaire)




```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

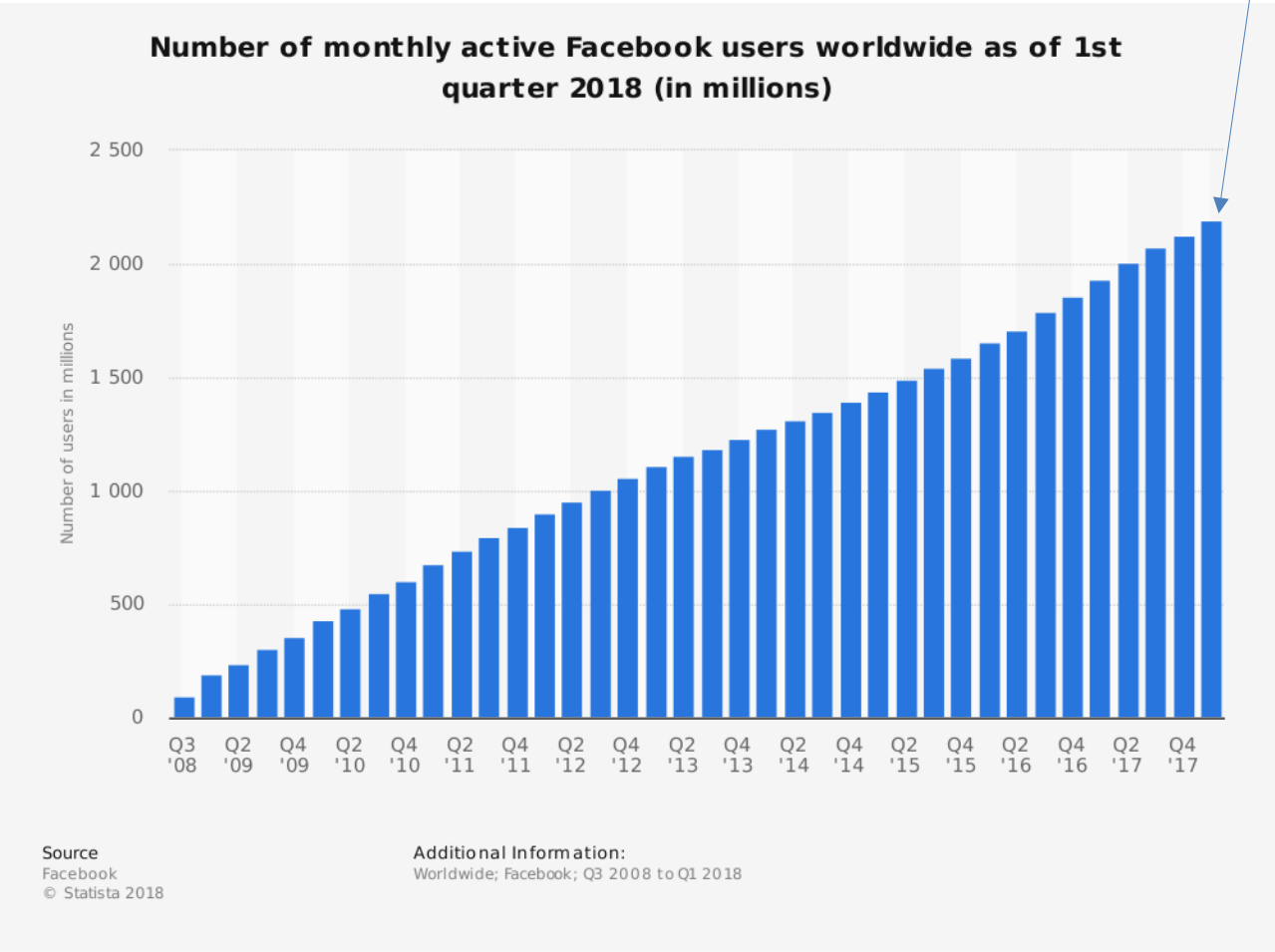
Recherche Séquentielle (Linéaire)




```
def linear_search ( name, list_of_names ):
  for list_name in list_of_names:
    if name == list_name:
      return True
  return False
```

Facebook

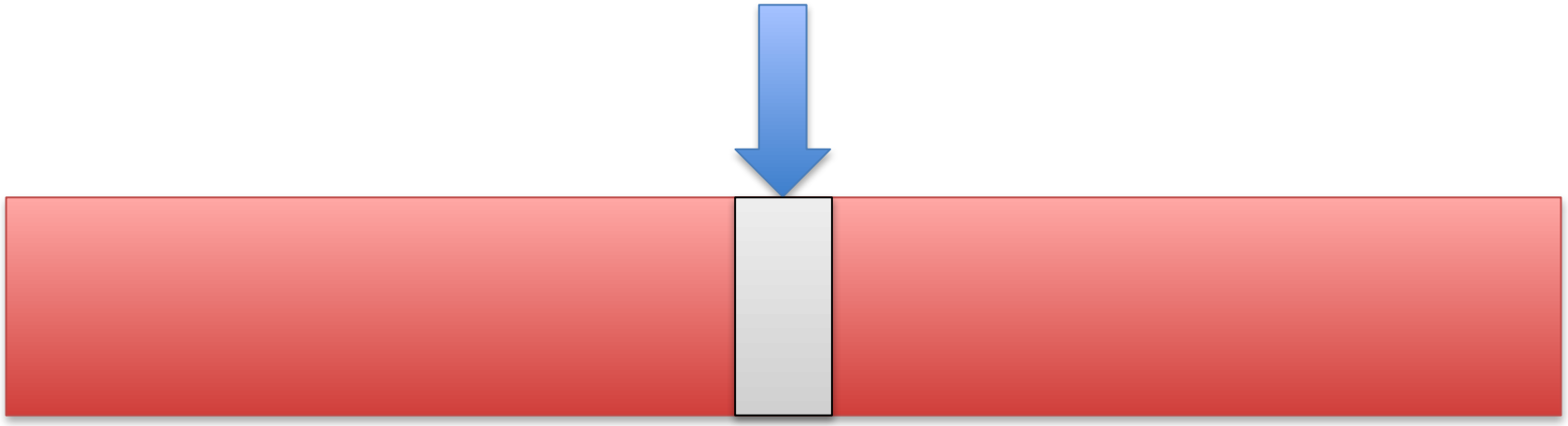
2.23 milliard!!!



Comment on cherche dans un dictionnaire?

Recherche dichotomique

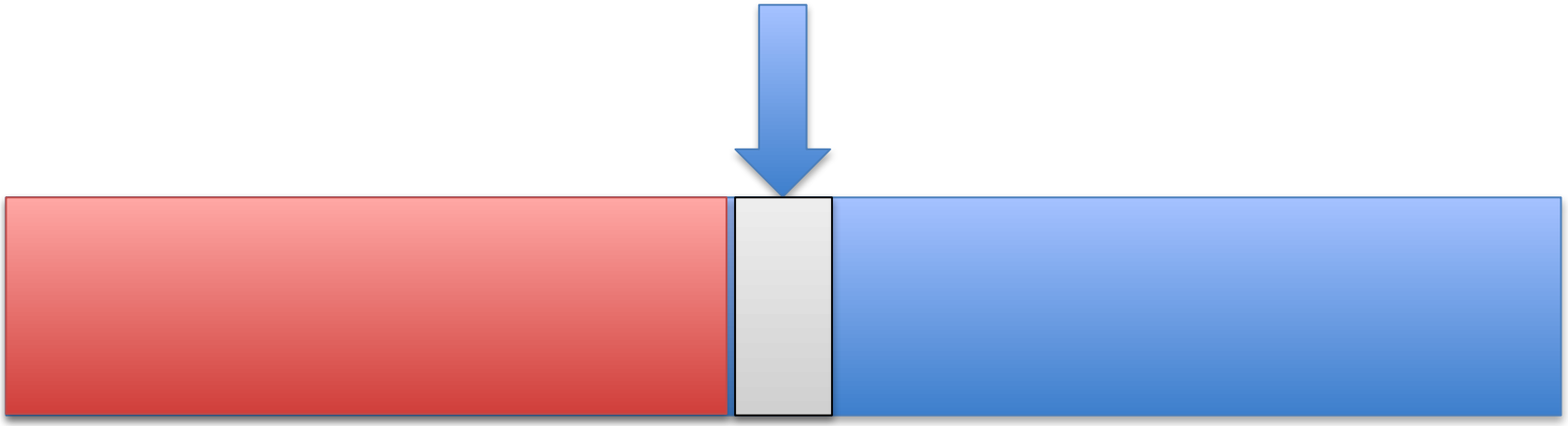
- Algorithme pour les **listes ordonnées**



- On commence au milieu
- Est-ce que le mot cherché est là?
  - Si oui: arrêtez
  - Si non...

Recherche dichotomique

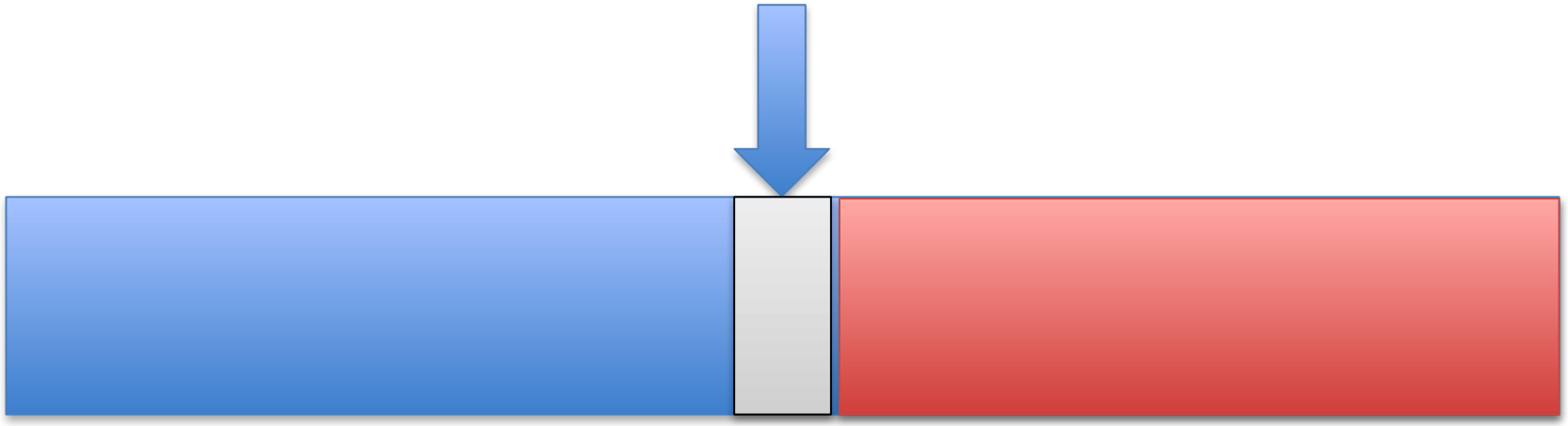
- Si le mot n'est pas au milieu



- Est-ce que le mot cherché est inférieur dans l'ordre alphabétique?
  - Si oui, continuez à gauche

Recherche dichotomique

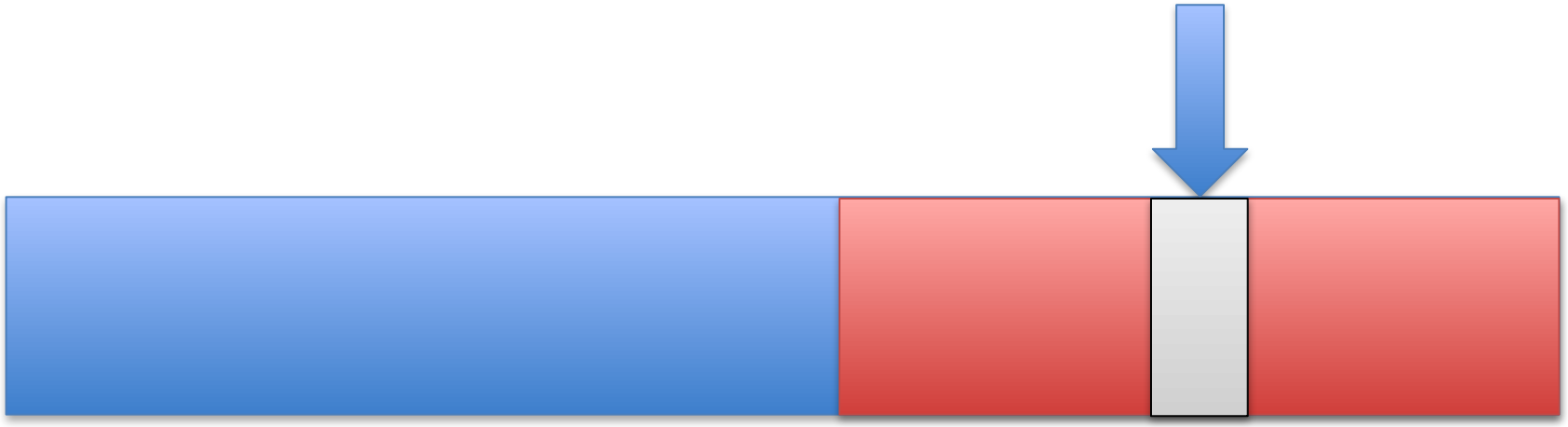
- Si le mot n'est pas au milieu



- Est-ce que le mot cherché est inférieur dans l'ordre alphabétique?
  - Si non, continuez à droite

Recherche dichotomique

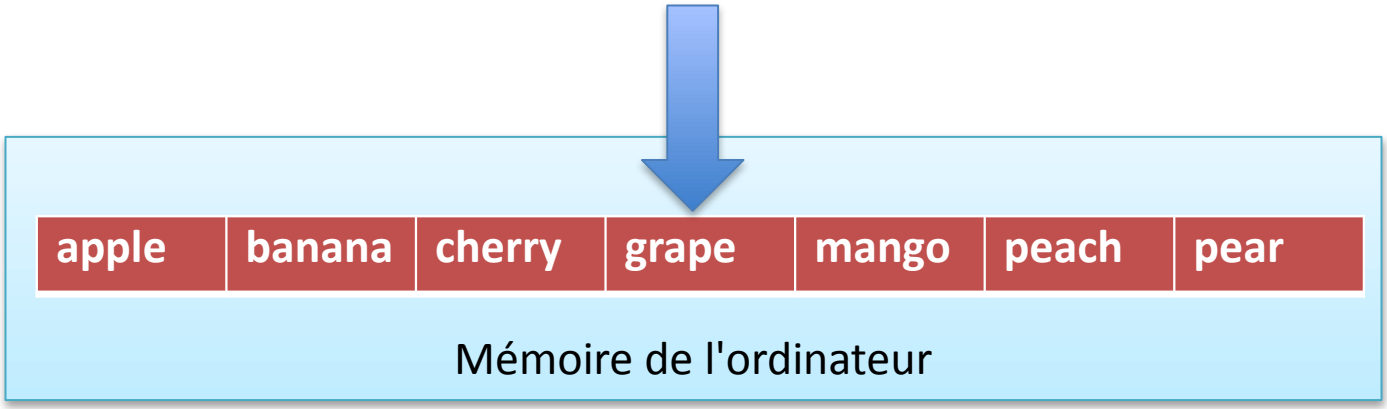
- On continue là de la même manière



On considère le milieu de la partie droite

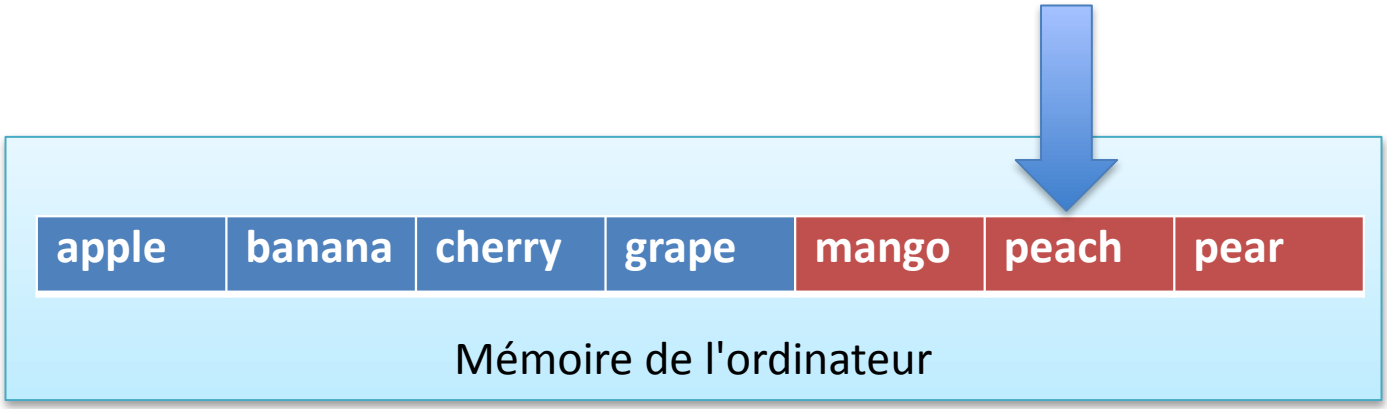
- On ne regarde jamais le dictionnaire complet!
- Pour une liste de *n* éléments, on regarde log *n* éléments (Pour Facebook, 31 noms!)

Recherche dichotomique



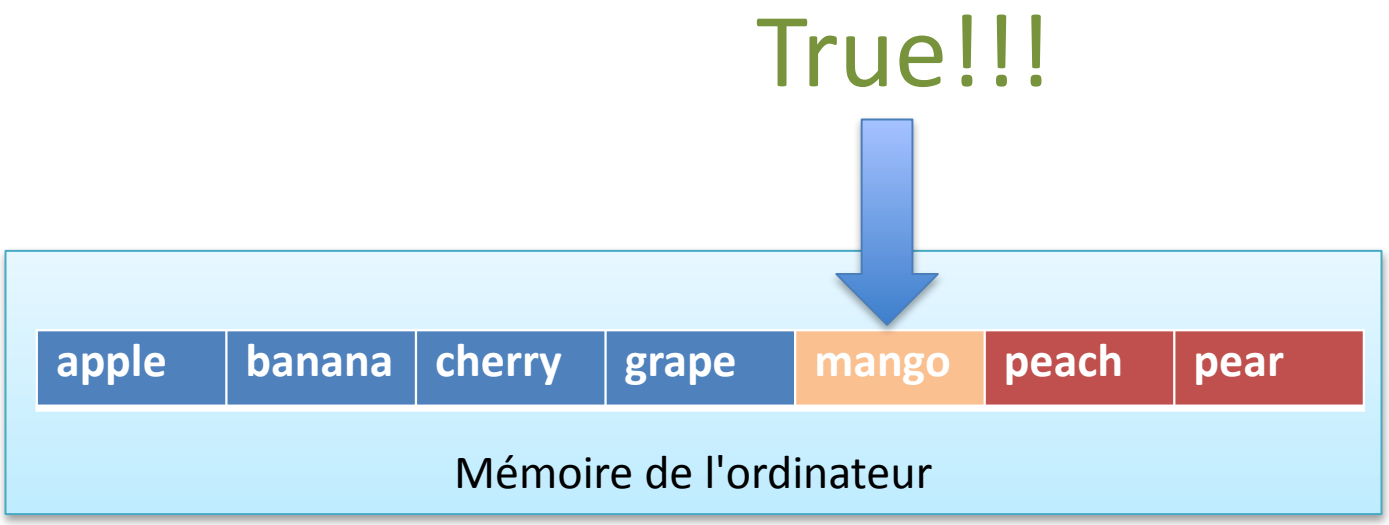
- On cherche "mango"

Recherche dichotomique



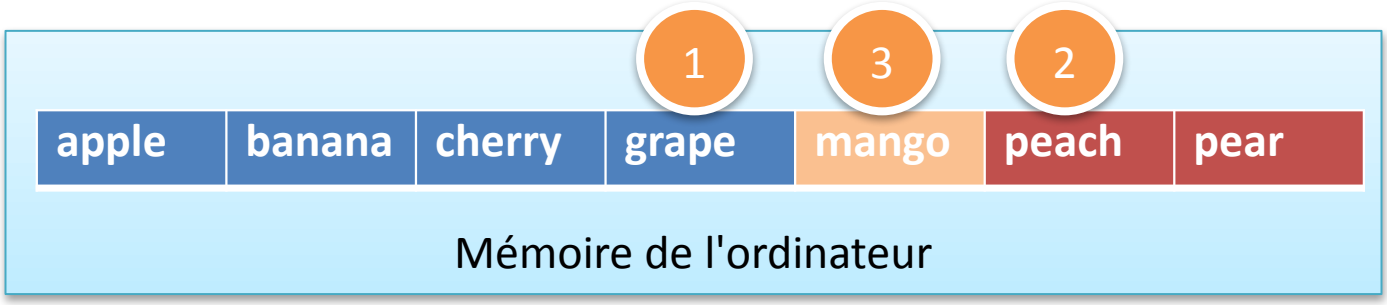
- On cherche "mango"

Recherche dichotomique



- On cherche "mango"

Recherche dichotomique



- On cherche "mango"

Recherche dichotomique

- Pendant la mission vous allez étudier:

```
def binary_search ( name, list_of_names ):
  first = 0
  last = len(list_of_names)-1
  found = False
  while first<=last and not found:
    middle = (first + last)//2
    if list_of_names[middle] == name:
      found = True
    else:
      if name < list_of_names[middle]:
        last = middle-1
      else:
        first = middle+1
  return found
```