Informatique 1
Introduction à la programmation

Mission 5 : restructuration

Kim Mens   Siegfried Nijssen   Charles Pecheur

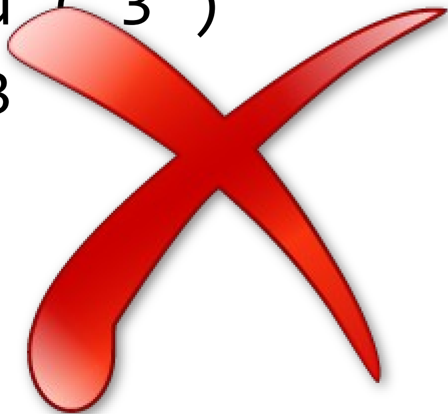
Agenda

- Restructuration
  - Spécifications et Tests
  - Tuples
  - Algorithmes de recherche

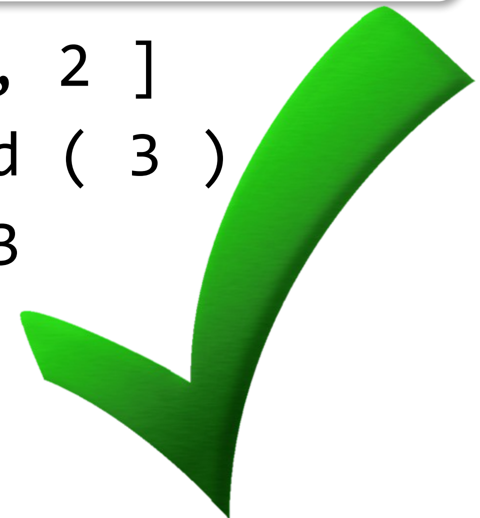
Tuples

Tuples	Listes
Tuples ne sont **pas** mutables	Listes sont mutables

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



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

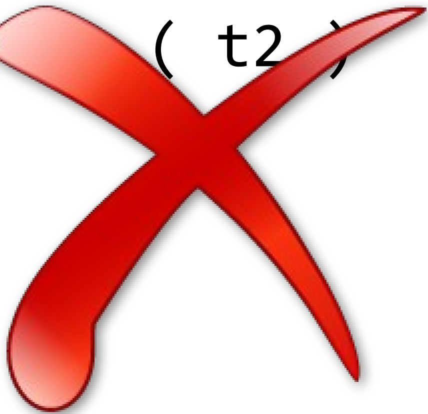


Tuples

```
t = ( 2, 4 )
t2 = ( 2, 3 ) + t
print ( t2 )
```

Tuples

```
t = ( 2 )
t2 = ( 2, 3 ) + t
pr      ( t2 )
```

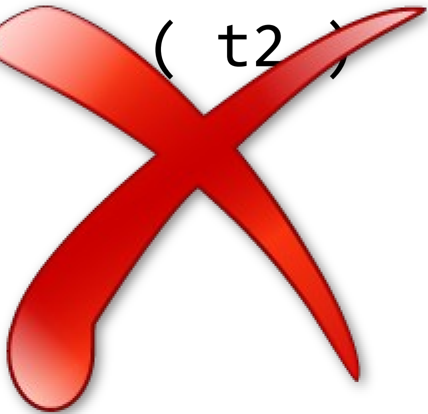


Tuples

```
t = ( 2, )
t2 = ( 2, 3 ) + t
print ( t2 )

(2,3,2)
```

Tuples

```
t = [ 2 ]
t2 = ( 2, 3 ) + t
pr    ( t2 )
```



Tuples

```
t = [ 2 ]
t2 = ( 2, 3 ) + tuple(t)
print ( t2 )
```

Tuples

```
l = [(1,2),(3,4),(5,6)]
for t in l:
    print (t[0],t[1])
```

```
l = [(1,2),(3,4),(5,6)]
for x,y in l:
    print (x,y)
```

Spécifications et Tests

```
def ordre ( l ):
  """ Détermine si l est une liste ordonnée

      Une liste est ordonnée
      - si l[i] <= l[i+1] pour tout i
      - ou si l[i] >= l[i+1] pour tout i
      Par exemple, [0,1,1,2] est ordonnée,
      [2,1,1,0] est ordonnée, mais
      [2,0,1,1] n'est pas ordonnée.

      Args:
         l: une liste de nombres entiers
      Returns:
         Un booléen
         True s'il y a un ordre
         False s'il n'y pas d'ordre
  """
```

Spécifications et Tests

```
def test_ordre ():

  assert ordre ( [ 0, 1, 1, 2 ] ) == True
  assert ordre ( [ 2, 1, 1, 0 ] ) == True
  assert ordre ( [ 2, 0, 1, 1 ] ) == False
```

Spécifications et Tests

```
def test_ordre ():

  assert ordre ( [ 0, 1, 1, 2 ] ) == True
  assert ordre ( [ 2, 1, 1, 0 ] ) == True
  assert ordre ( [ 2, 0, 1, 1 ] ) == False
  assert ordre ( [ 1, 1 ] ) == True
```

Spécifications et Tests

```
def test_ordre ():

  assert ordre ( [ 0, 1, 1, 2 ] ) == True
  assert ordre ( [ 2, 1, 1, 0 ] ) == True
  assert ordre ( [ 2, 0, 1, 1 ] ) == False
  assert ordre ( [ 1, 1 ] ) == True
  assert ordre ( [ 0, 1, 2, 3, 2 ] ) == False
  assert ordre ( [ 4, 3, 2, 1, 2 ] ) == False
  assert ordre ( [ 0, 1, 2, 5 ] ) == True
```

Spécifications et Tests

- "Pseudo-code": un algorithme en langage *presque naturel*, sans référence à un langage de programmation en particulier.

```
fonction ordre ( l )
  Si l'ordre de l est croissant, retourner Vrai
  Si l'ordre de l est décroissant, retourner Vrai
  retourner Faux
Fin de fonction
```

```
fonction est croissant ( l )
  pour tout index i dans l
    si l[i] > l[i+1], retourner Faux
  Fin pour
  retourner Vrai
Fin de fonction
```

Spécifications et Tests

```
def est_croissant ( l ):
    for i in range(len(l)):
        if l[i] > l[i+1]:
            return False
    return True

def est_decroissant ( l ):
    for i in range(len(l)):
        if l[i] < l[i+1]:
            return False
    return True

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

(1) séquence (indice)
(2) Condition de traitement
(3) Corps de la boucle

Spécifications et Tests

```
def est_croissant ( l ):
    for i in range(len(l)-1):
        if l[i] > l[i+1]:
            return False
    return True

def est_decroissant ( l ):
    for i in range(len(l)-1):
        if l[i] < l[i+1]:
            return False
    return True

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

Spécifications et Tests

```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

```
assert ordre ( [ 0, 1, 1, 2 ] ) == True
```

Spécifications et Tests

```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1
```

```
l = [ 0, 1, 1, 2 ]

l = [ 1, 1, 2 ]

l = [ 1, 2 ]

l = [ 2 ]

return True
```

Spécifications et Tests

```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

```
assert ordre ( [ 0, 1, 1, 2 ] ) == True
```

Spécifications et Tests

```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

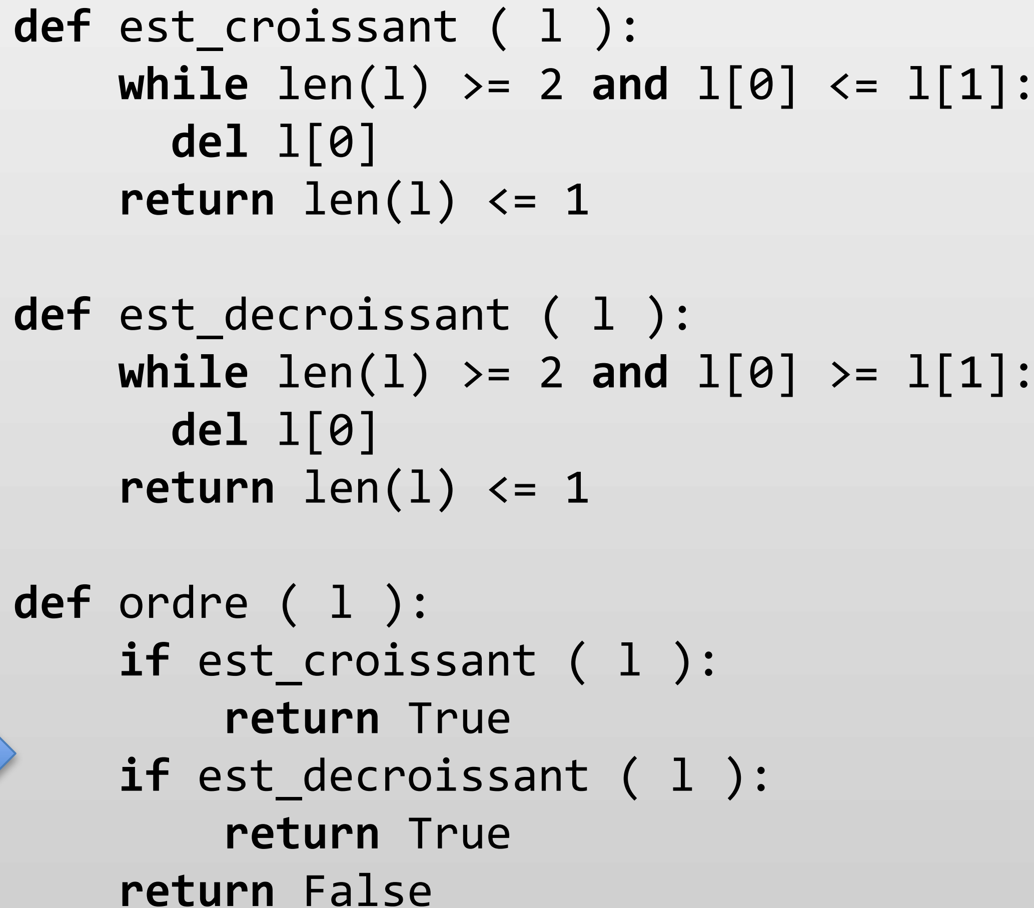
```
assert ordre ( [ 0, 1, 2, 3, 2 ] ) == False
```

Spécifications et Tests

```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1
```

```
l = [ 0, 1, 2, 3, 2 ]

l = [ 1, 2, 3, 2 ]

l = [ 2, 3, 2 ]

l = [ 3, 2 ]

return False
```

Spécifications et Tests



```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

```
assert ordre ( [ 0, 1, 2, 3, 2 ] ) == False
```

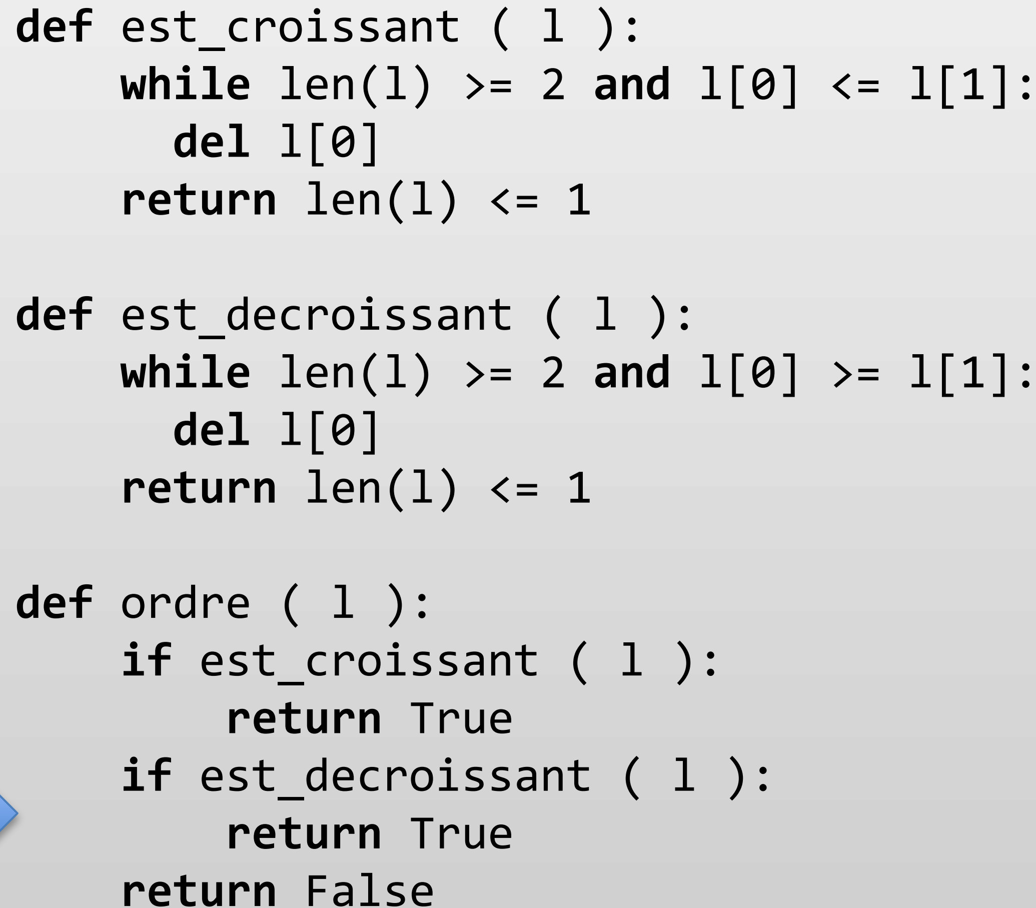
Spécifications et Tests

```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

l = [3,2]

```
assert ordre ( [ 0, 1, 2, 3, 2 ] ) == False
```

Spécifications et Tests



```
def est_croissant ( l ):
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

```
assert ordre ( [ 0, 1, 2, 3, 2 ] ) == False
```

Spécifications et Tests

Sauf si il est explicitement spécifié, une fonction NE DOIT JAMAIS modifier une liste passée en argument

Spécifications et Tests

```
def est_croissant ( l ):
    l = list ( l )
    while len(l) >= 2 and l[0] <= l[1]:
      del l[0]
    return len(l) <= 1

def est_decroissant ( l ):
    l = list ( l )
    while len(l) >= 2 and l[0] >= l[1]:
      del l[0]
    return len(l) <= 1

def ordre ( l ):
    if est_croissant ( l ):
        return True
    if est_decroissant ( l ):
        return True
    return False
```

La solution sans **del** est préférée!

Recherche Linéaire

```
def est_croissant ( l ):
    for i in range(len(l)-1):
        if l[i] > l[i+1]:
            return False
    return True
```

```
def est_dans( c, l ):
    for car in l:
        if car == c:
            return True
    return False
```

```
    [Boucle for]
        if [Condition]:
            return [Résultat]
    return [Pas de résultat]
```

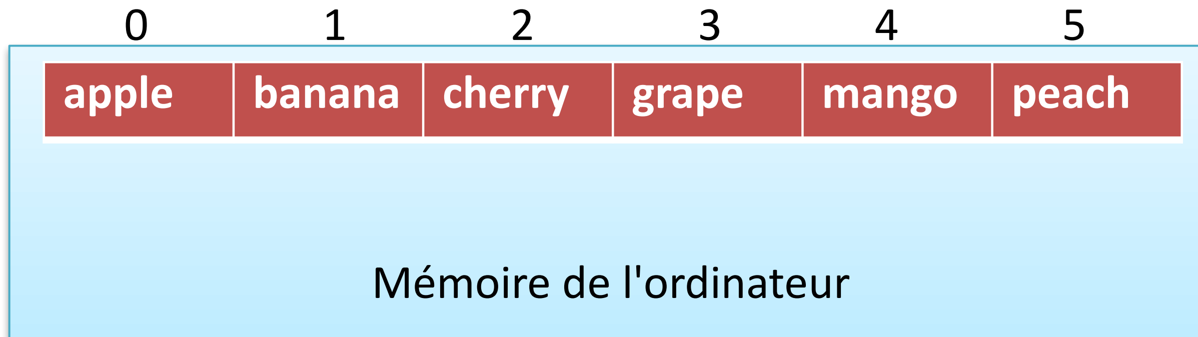
Recherche Linéaire

```
def est_dans( c, l ):
    for car in l:
        if car == c:
            return True
        else:
            return False
    return False
```

Recherche dichotomique

```
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
```

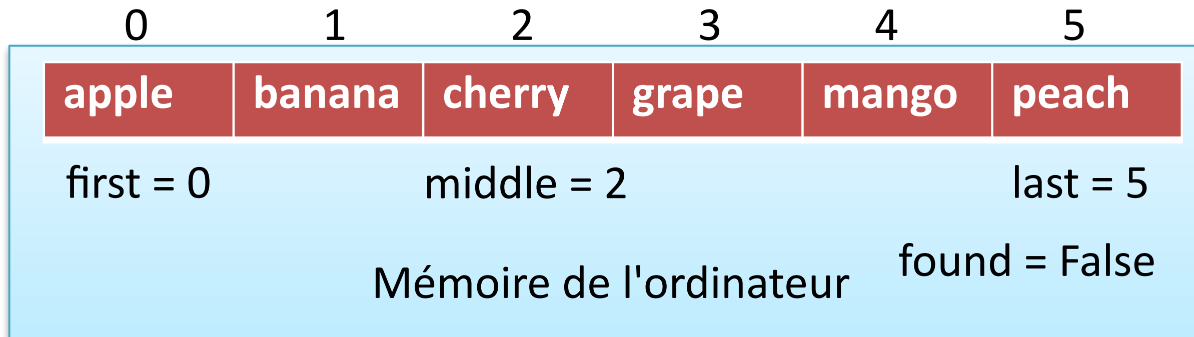
Recherche dichotomique



Cherchons "banana"

```
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
```

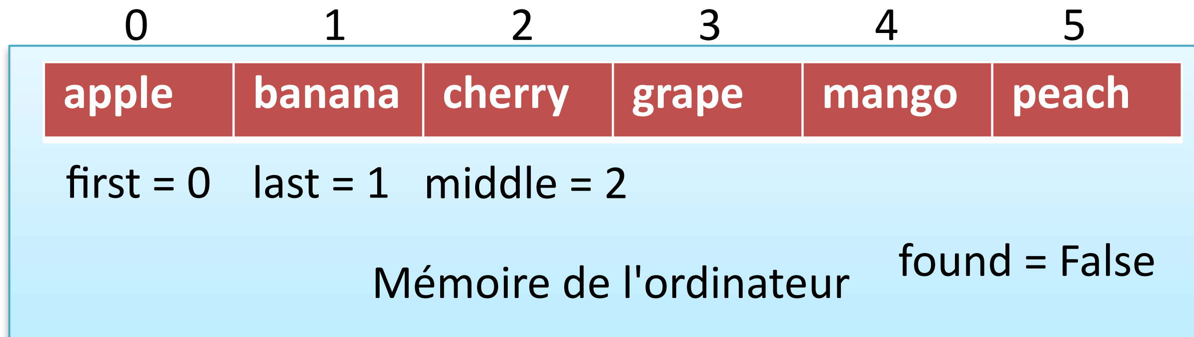
Recherche dichotomique



Cherchons "banana"

```
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
```

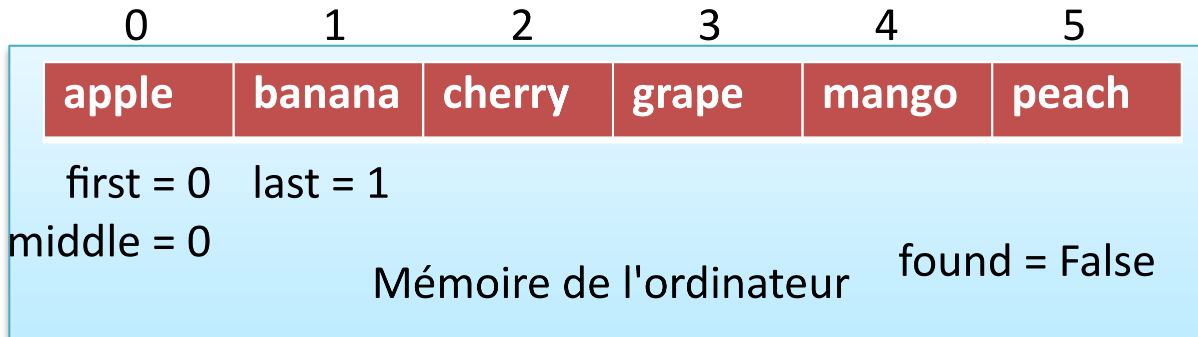
Recherche dichotomique



Cherchons "banana"

```
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
```

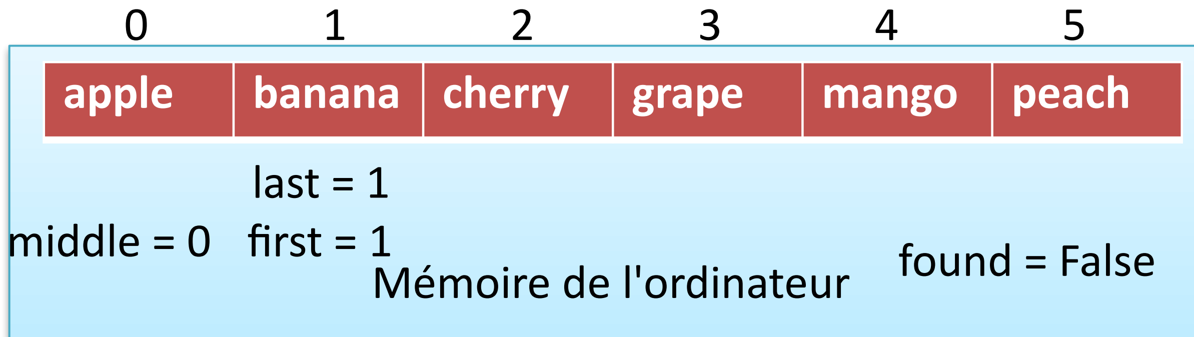
Recherche dichotomique



Cherchons "banana"

```
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
```

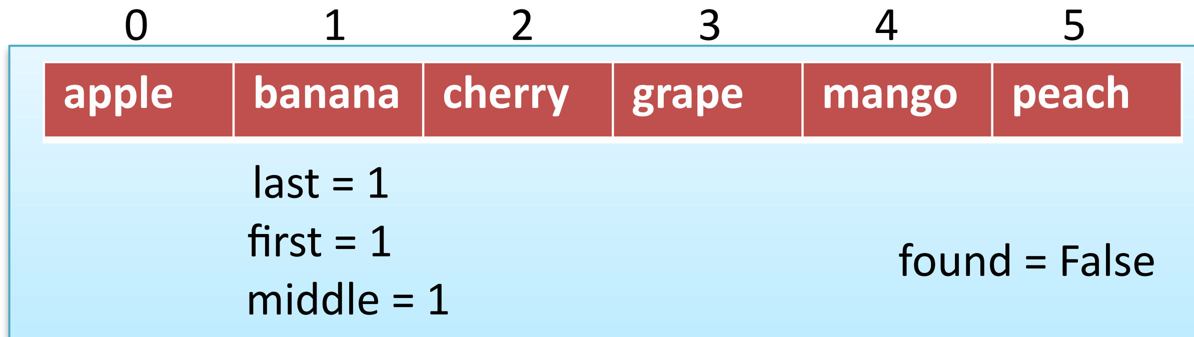
Recherche dichotomique



Cherchons "banana"

```
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
```

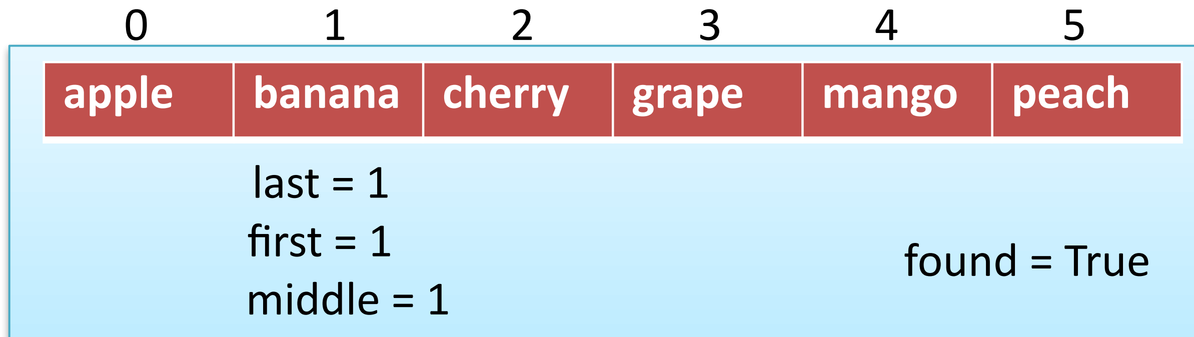
Recherche dichotomique



Cherchons "banana"

```
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
```

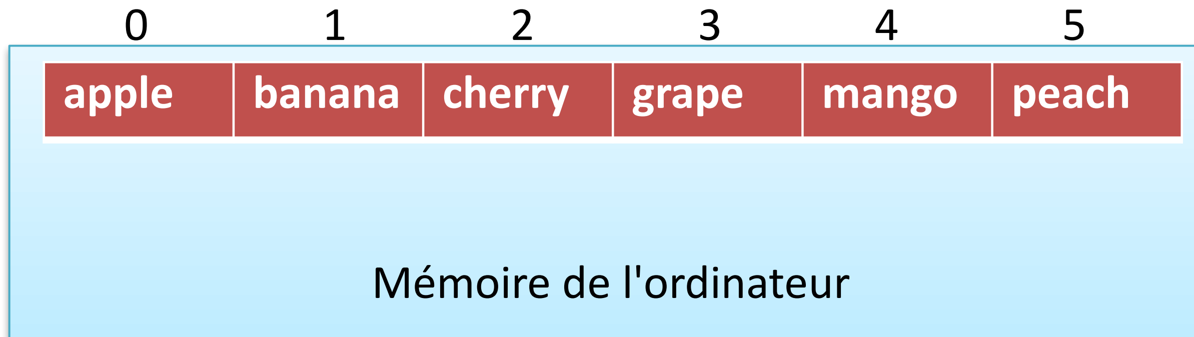
Recherche dichotomique



Cherchons "banana"

```
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
```

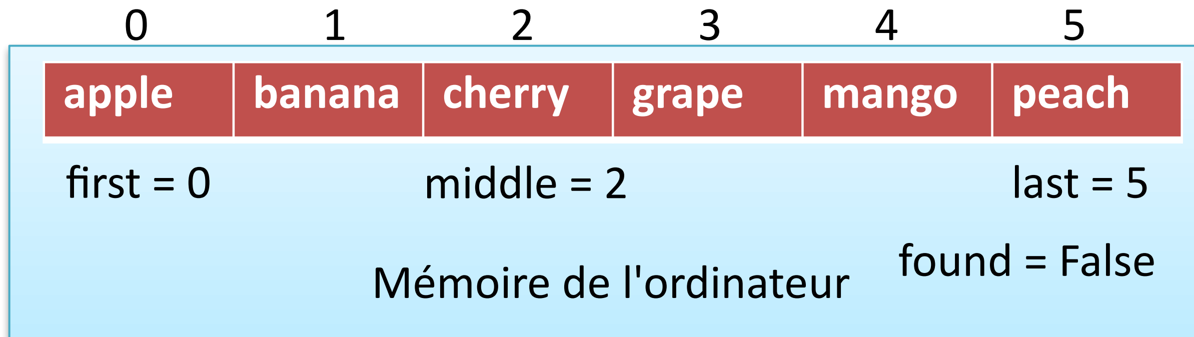
Recherche dichotomique



Cherchons "bobo"

```
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
```

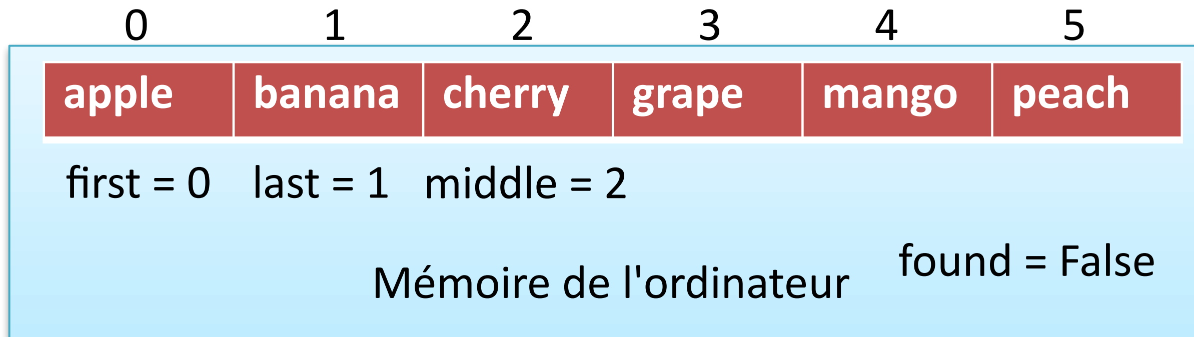
Recherche dichotomique



Cherchons "bobo"

```
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
```

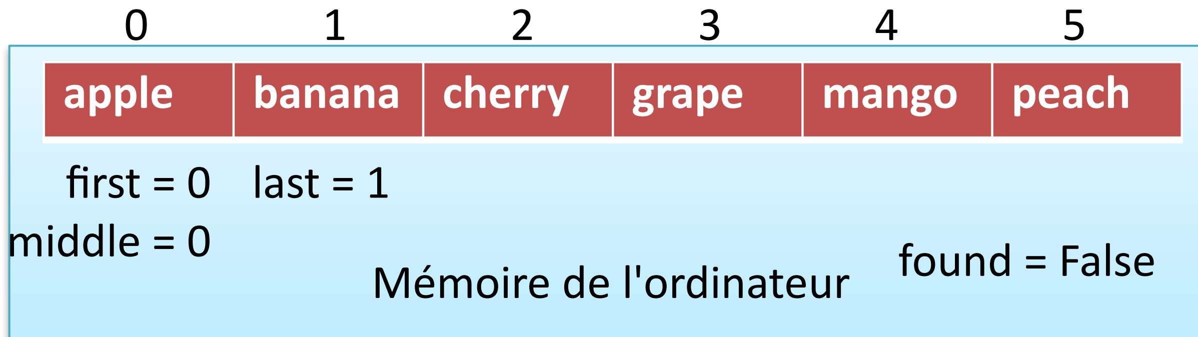
Recherche dichotomique



Cherchons "bobo"

```
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
```

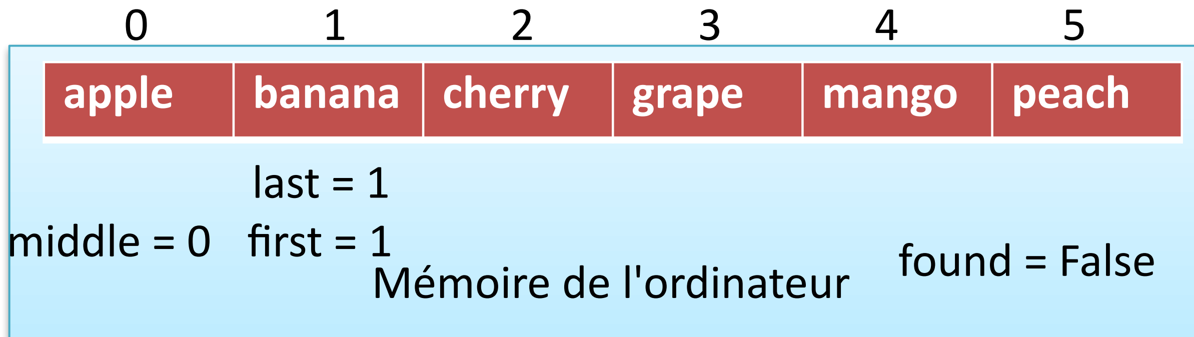
Recherche dichotomique



Cherchons "bobo"

```python
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
```

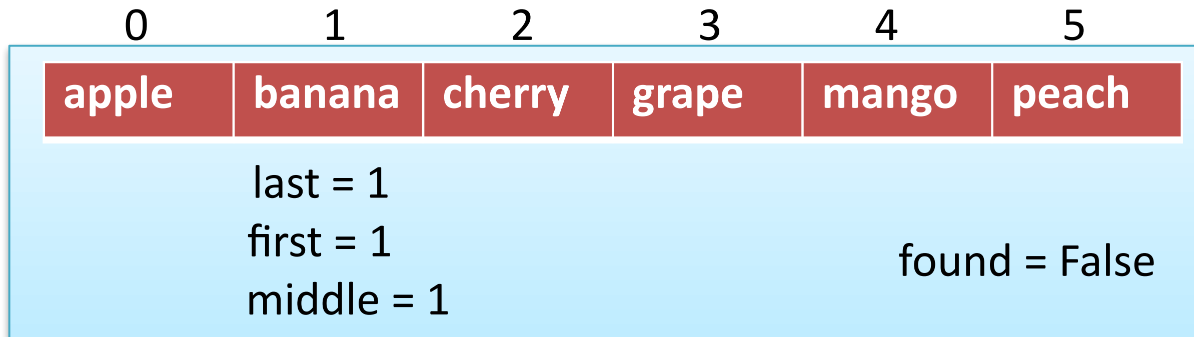
Recherche dichotomique



Cherchons "bobo"

```
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
```

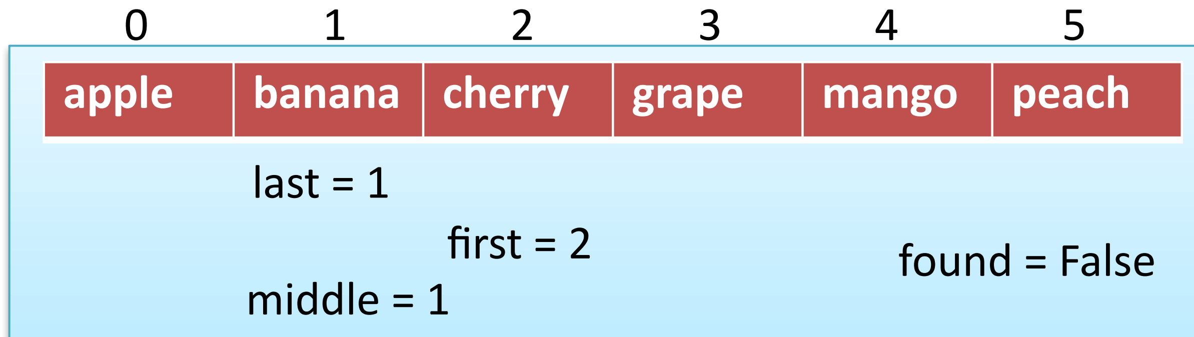
Recherche dichotomique



Cherchons "bobo"

```
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
```

Recherche dichotomique



Cherchons "bobo"

```
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
```