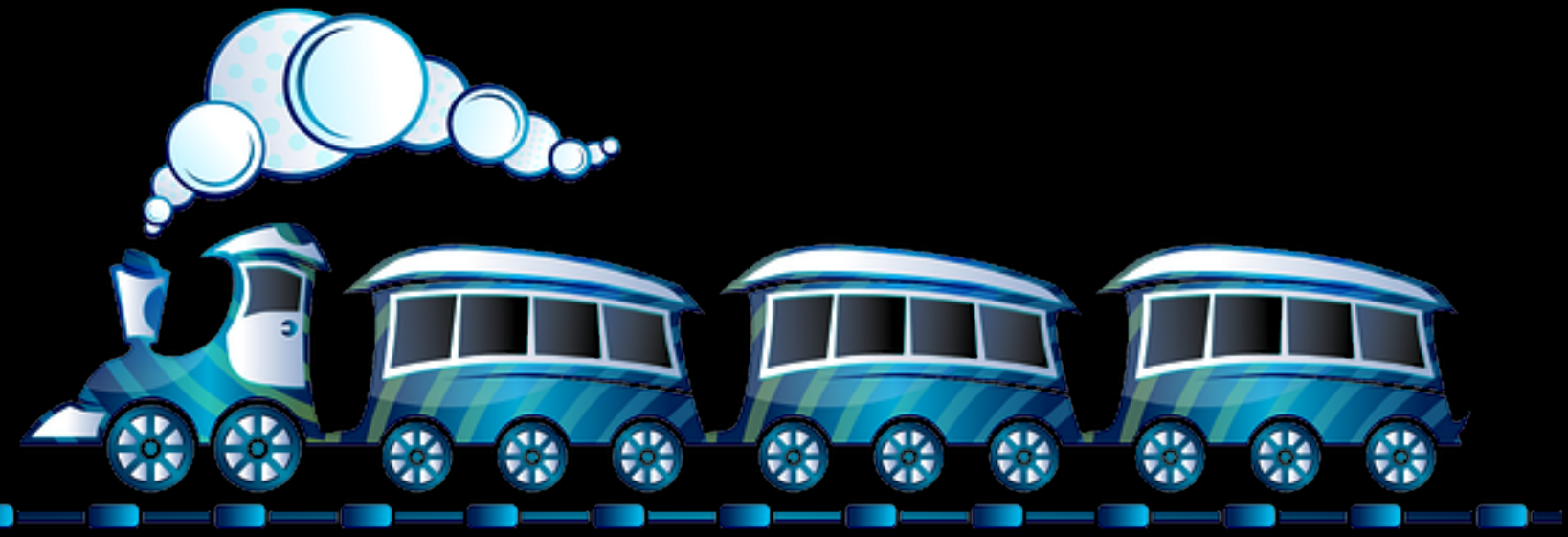
Terminale NSI

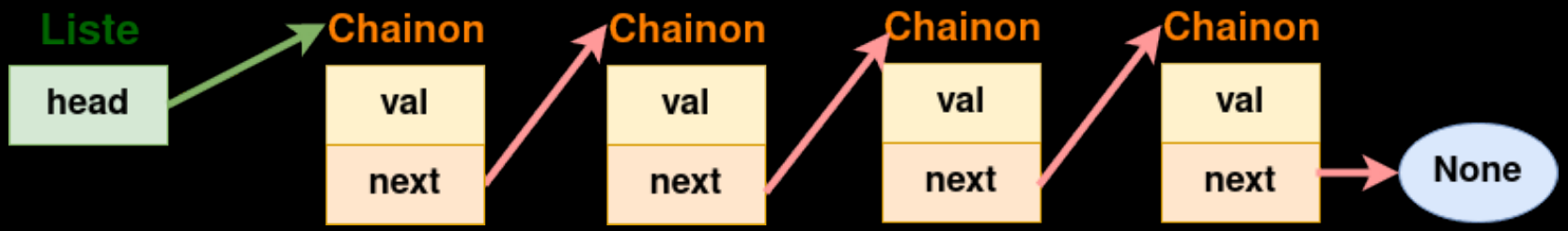


Les listes chaînées

M. Tonnelier 2024

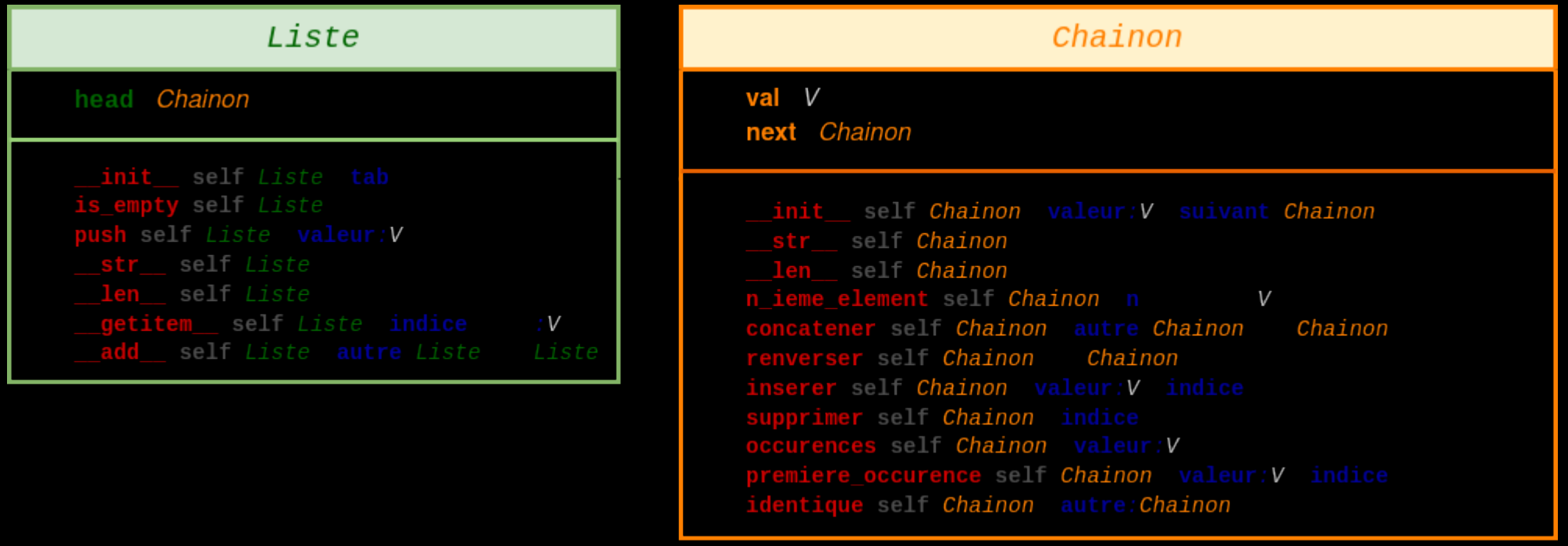
Un objet Liste = une instance de la classe Liste

Schéma de la représentation en mémoire :



Implantation d'une liste chaînée par **récursivité**

Diagramme des classes :



La classe Chainon
= un maillon de la chaîne

Attributs :

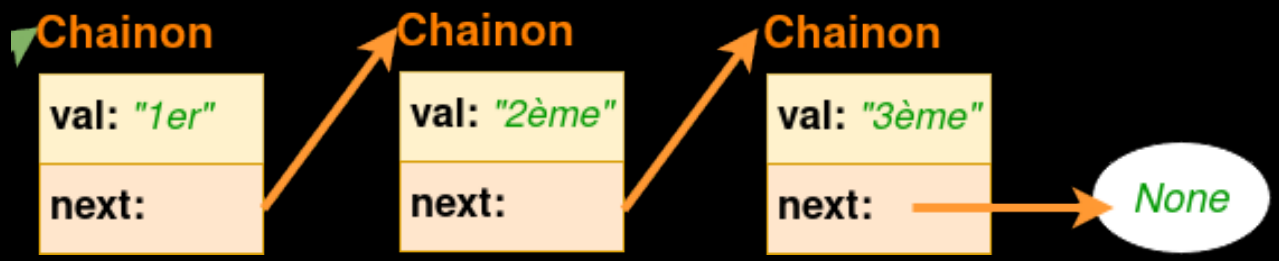
Méthodes :

Chainon

val *V*
next *Chainon*

```
__init__ self Chainon  valeur:V  suivant Chainon
__str__ self Chainon
__len__ self Chainon
n_ieme_element self Chainon  n          V
concatener self Chainon  autre Chainon    Chainon
renverser self Chainon    Chainon
inserer self Chainon  valeur:V  indice
supprimer self Chainon  indice
occurences self Chainon  valeur:V
premiere_occurence self Chainon  valeur:V  indice
identique self Chainon  autre:Chainon
```

Un objet Chainon = une instance de cette classe

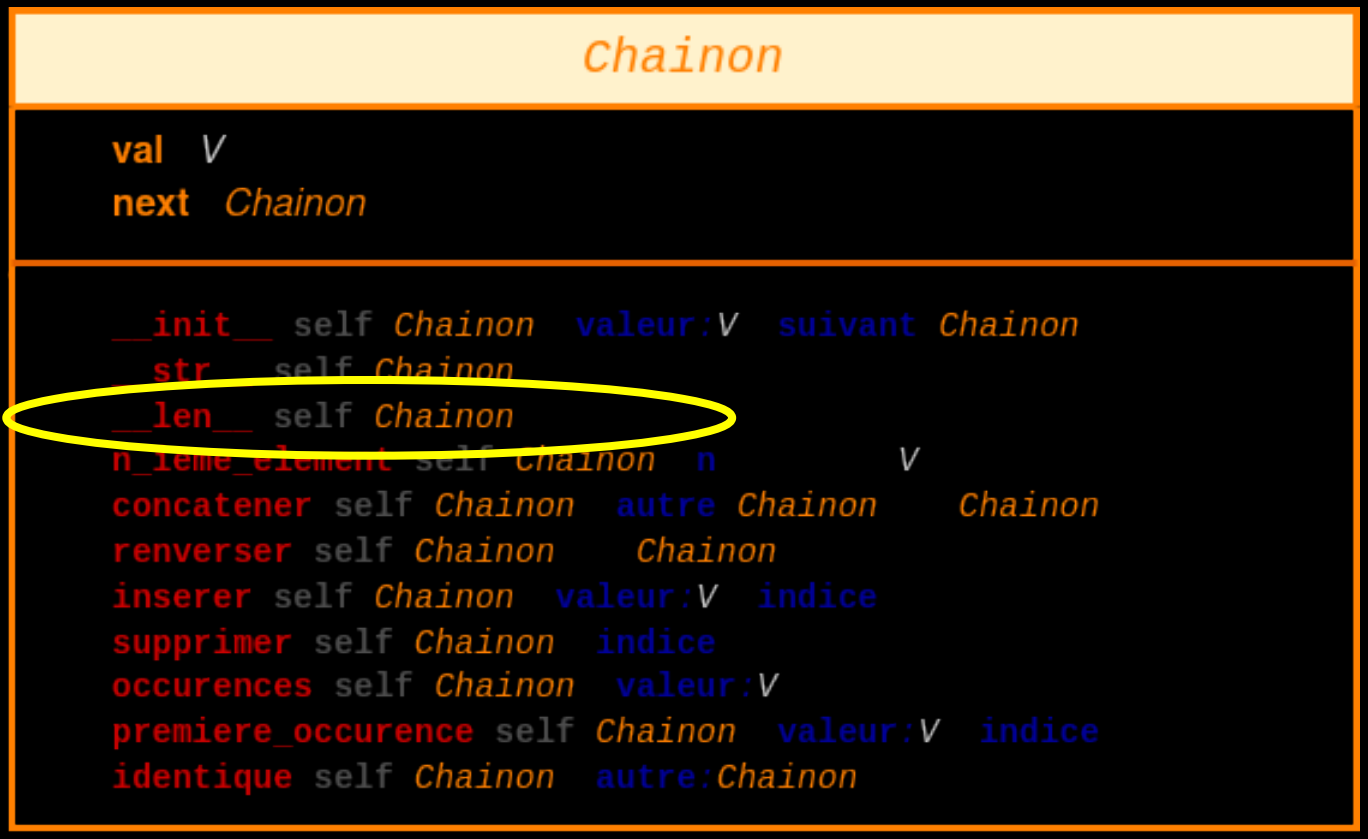



```
chaine = Chainon("1er", Chainon("2ème", Chainon("3ème")))
assert str(chaine) == "1er -> 2ème -> 3ème -> None"
```

La classe Chainon
= un maillon de la chaîne

Attributs :

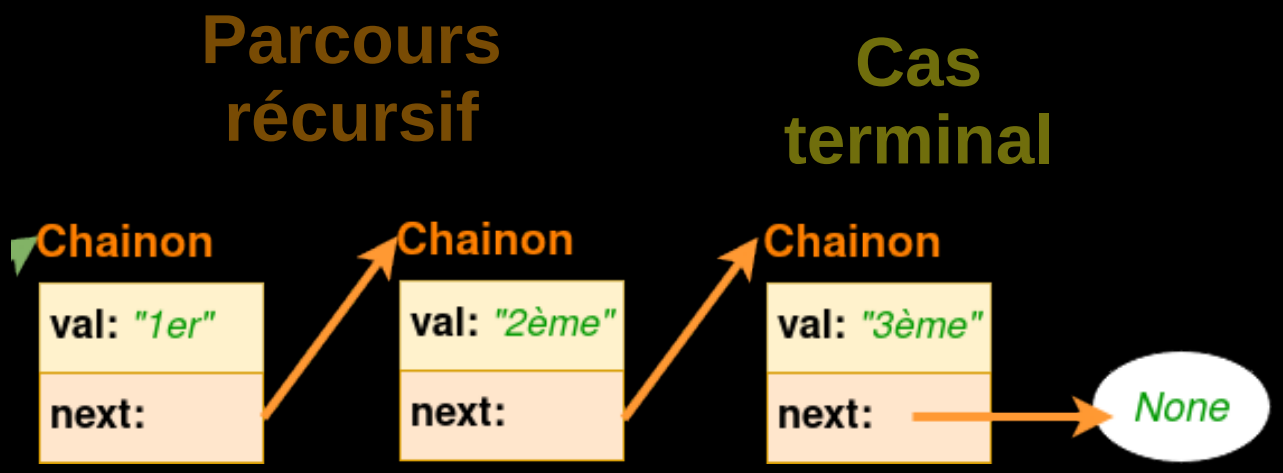
Méthodes :



Zoom sur la méthode len

Un objet Chainon

longueur d'un maillon : 2 cas

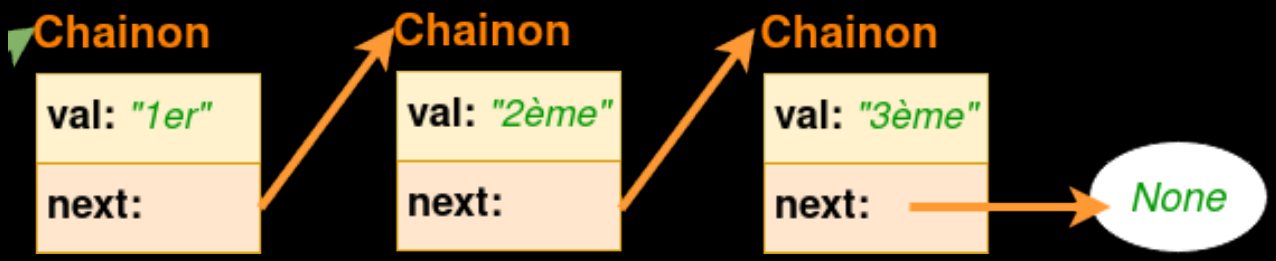


Cas récursif :
Il faut **avancer**
pour trouver
le dernier maillon

Cas terminal :
C'est **le dernier**
maillon

Longueur d'un Chainon

```
chaine = Chainon("1er", Chainon("2ème", Chainon("3ème")))
print(chaine)
print("longueur =", len(chaine))
```




```
1er -> 2ème -> 3ème -> None
longueur = 3
```

Longueur d'un Chainon

```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```

APPEL 1 : len(1er -> 2ème -> 3ème -> None)

1 + appel récursif sur le suivant

Longueur d'un Chainon

```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```

APPEL 1 : len(1er -> 2ème -> 3ème -> None)
1 + appel récursif sur le suivant

APPEL 2 : len(2ème -> 3ème -> None)
1 + appel récursif sur le suivant

Longueur d'un Chainon

```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```

APPEL 1 : len(1er -> 2ème -> 3ème -> None)
1 + appel récursif sur le suivant

APPEL 2 : len(2ème -> 3ème -> None)
1 + appel récursif sur le suivant

APPEL 3 : len(3ème -> None)
1 → cas terminal

Longueur d'un Chainon

```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```

APPEL 1 : len(1er -> 2ème -> 3ème -> None)
1 + appel récursif sur le suivant

APPEL 2 : len(2ème -> 3ème -> None)
1 + appel récursif sur le suivant

APPEL 3 : len(3ème -> None)
1 → cas terminal

1

Longueur d'un Chainon

```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```

APPEL 1 : len(1er -> 2ème -> 3ème -> None)

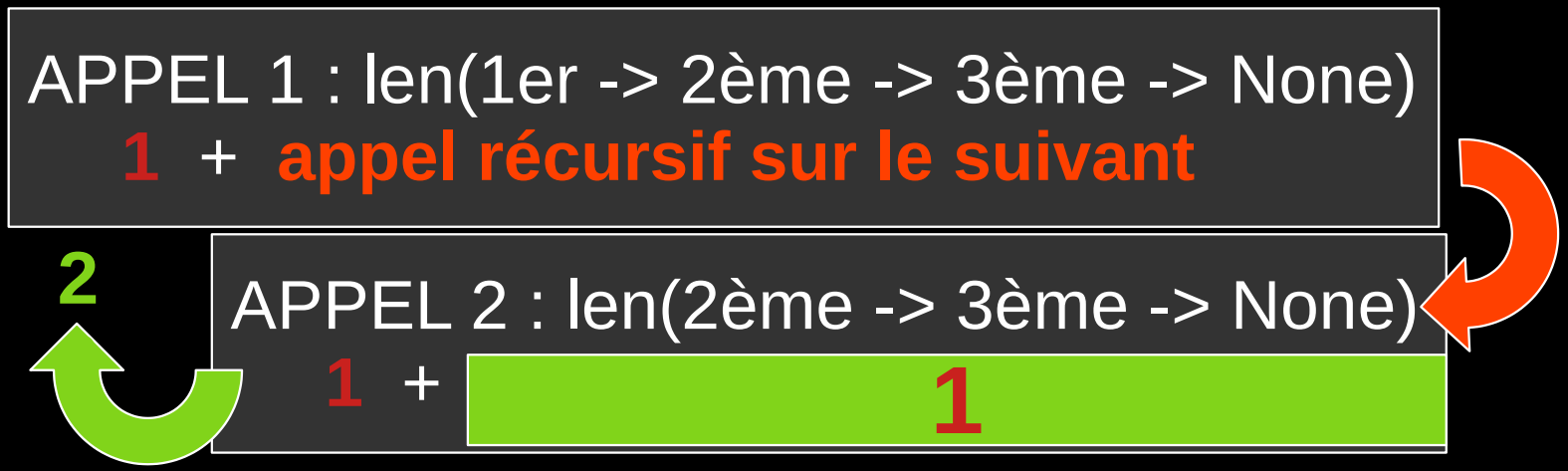
1 + **appel récursif sur le suivant**

APPEL 2 : len(2ème -> 3ème -> None)

1 + 1

Longueur d'un Chainon

```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```



Longueur d'un Chainon

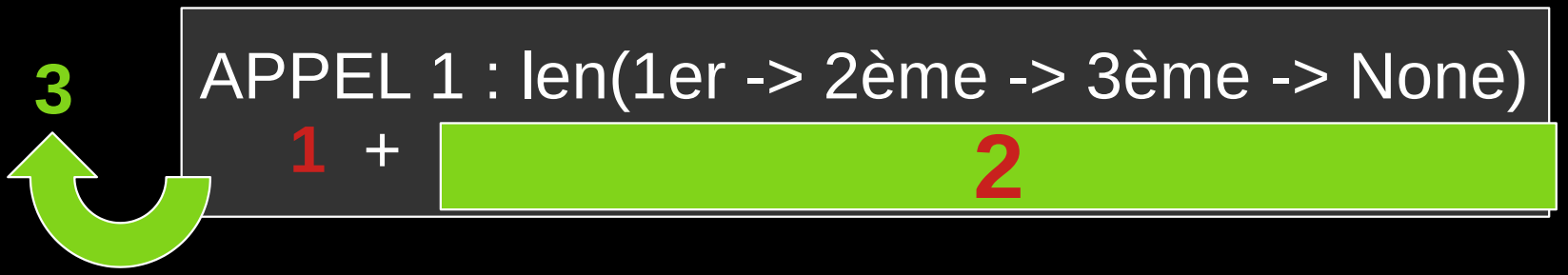
```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```

APPEL 1 : len(1er -> 2ème -> 3ème -> None)

1 + 2

Longueur d'un Chainon

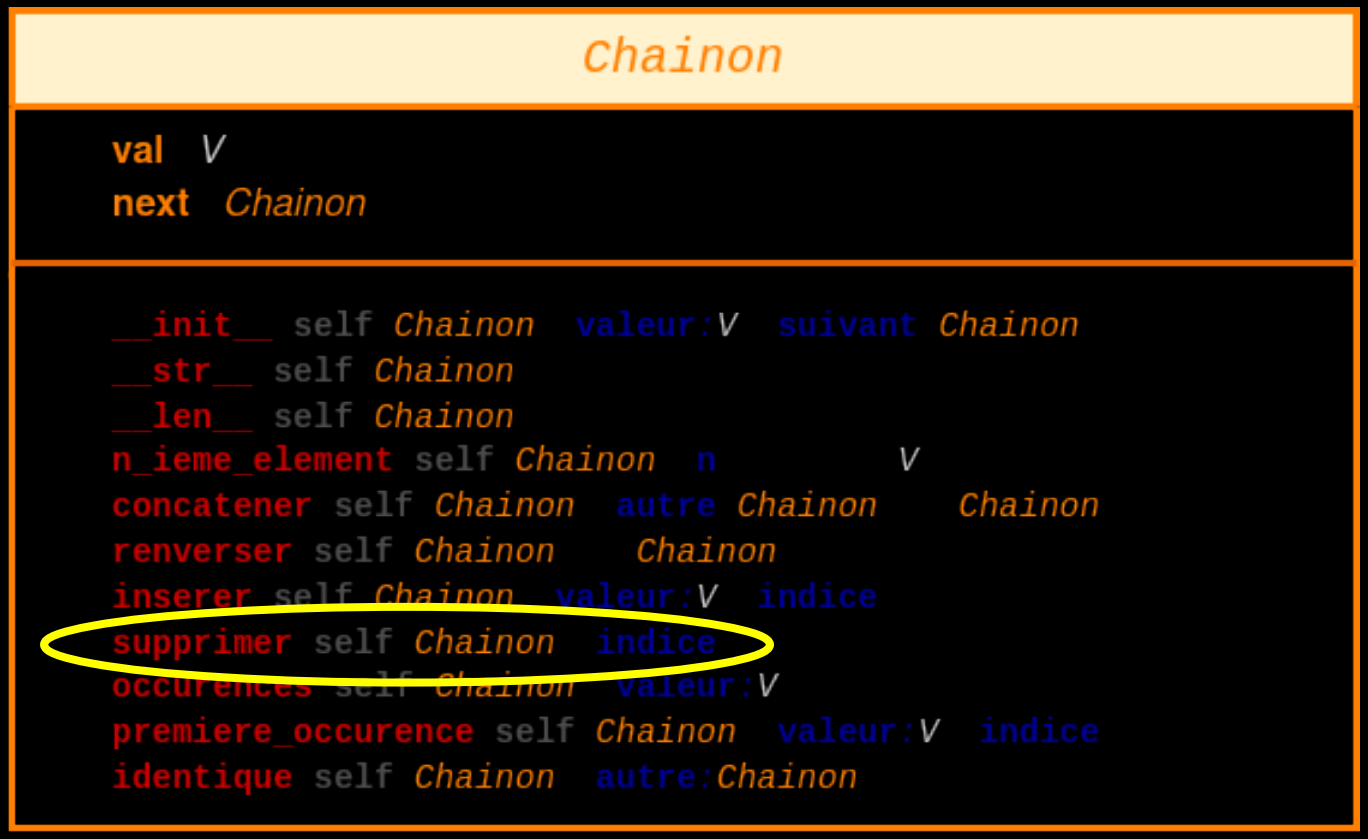
```
def __len__(self:C) ->int:
    """ Renvoie le nombre de Chainon de la chaîne """
    if self.next is None : # Cas terminal
        return 1
    else: # Cas récursif
        return 1 + len(self.next)
```



La classe Chainon
= un maillon de la chaîne

Attributs :

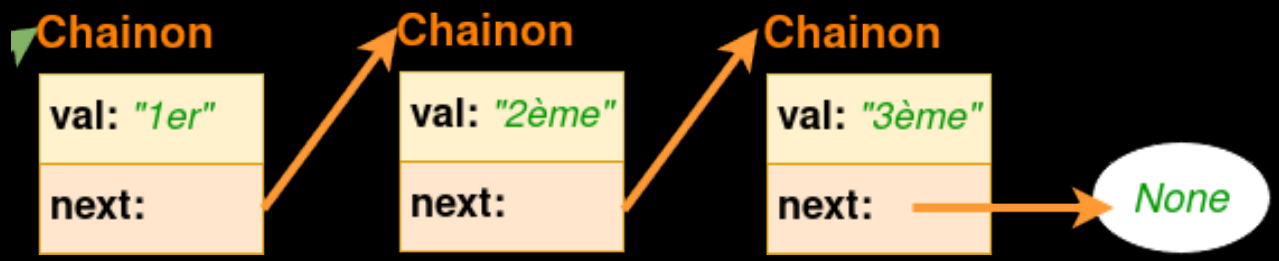
Méthodes :



Zoom sur la méthode supprimer

Un objet Chainon

supprimer un maillon : 4 cas



Cas terminal

Cas 1 :
C'est **le seul** maillon

Cas terminal

Cas 2 :
C'est le **premier** maillon

Cas terminal

Cas 3 :
Supprimer le maillon **suivant**

Parcours récursif

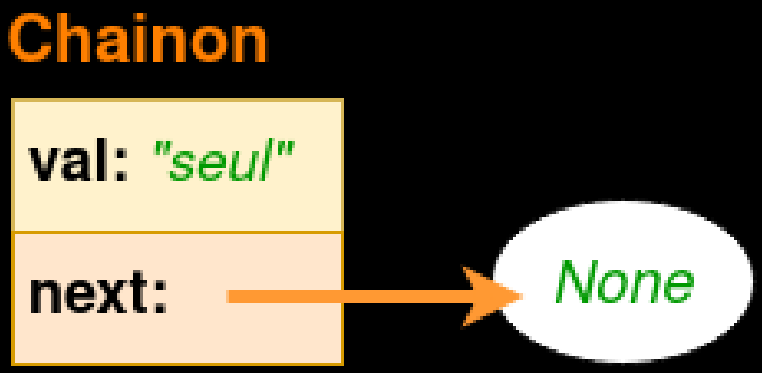
Cas 4 :
Il faut **avancer** pour trouver le maillon à supprimer

Supprimer un maillon : 4 cas

Cas terminal

Cas 1 :
C’est **le seul** maillon

Supprimer le seul maillon



On ne peut pas supprimer s'il y a un seul élément !

```
chaine = Chainon("seul")
try:
    chaine.supprimer(0)
except ValueError:
    assert True
else:
    assert False
```

```python
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste chainée. """
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
```

Cas terminal

Cas 1 :
C'est **le seul** maillon

On ne peut pas supprimer s'il y a un seul élément !

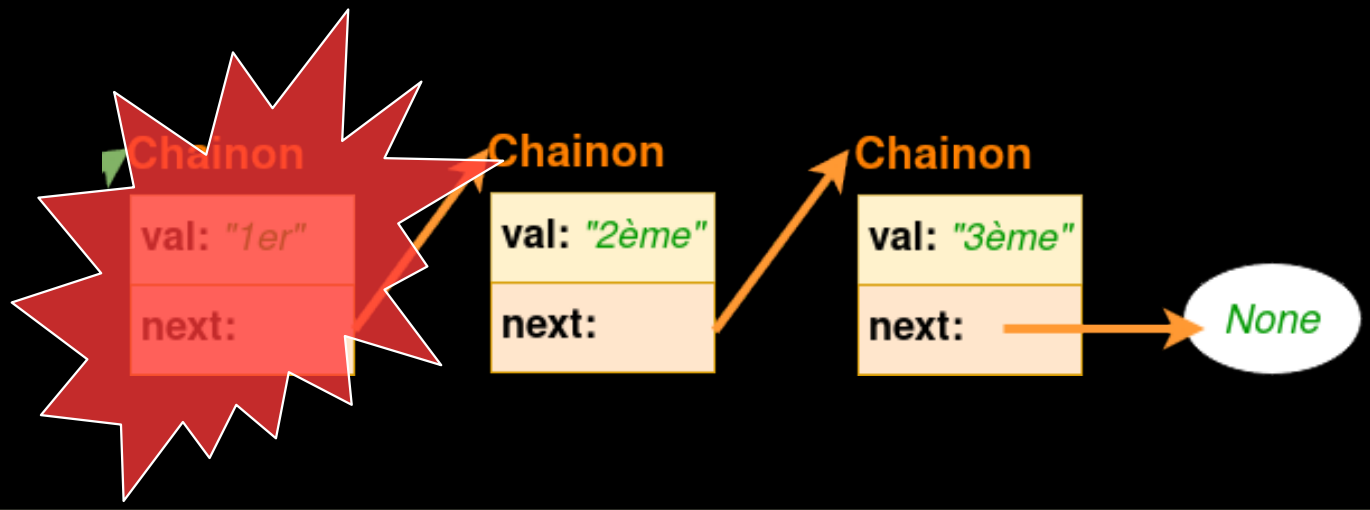
```python
chaine = Chainon("seul")
try:
    chaine.supprimer(0)
except ValueError:
    assert True
else:
    assert False
```

Supprimer un maillon : 4 cas

Cas terminal

Cas 2 :
C’est le
premier
maillon

Supprimer le premier




```
chaine = Chainon("1er", Chainon("2ème", Chainon("3ème")))
assert str(chaine) == "1er -> 2ème -> 3ème -> None"
chaine.supprimer(0)
assert str(chaine) == "2ème -> 3ème -> None"
```

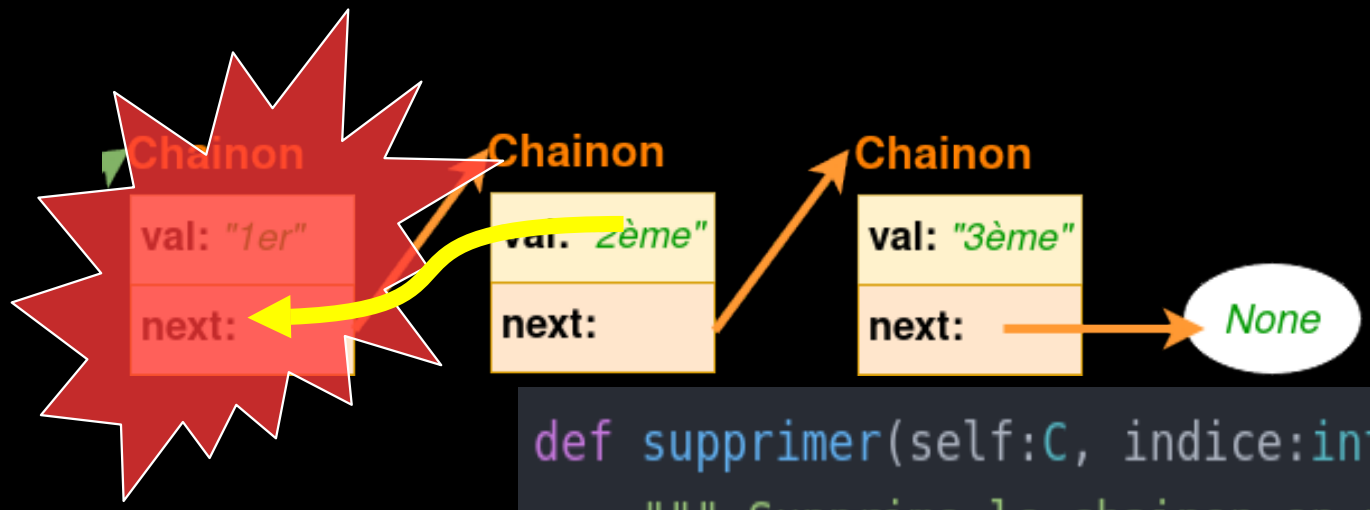
On ne peut pas supprimer le premier !

Supprimer le premier

On décale le second maillon dans le premier maillon

On ne peut pas supprimer le premier !

Supprimer le premier




```
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indic
    if self.longueur() == 1:
        raise ValueError("Impossible de suppr
    if indice == 0:
        self.val = self.next.val
        self.next = self.next.next
    elif indice == 1:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

On décale le second maillon dans le premier maillon

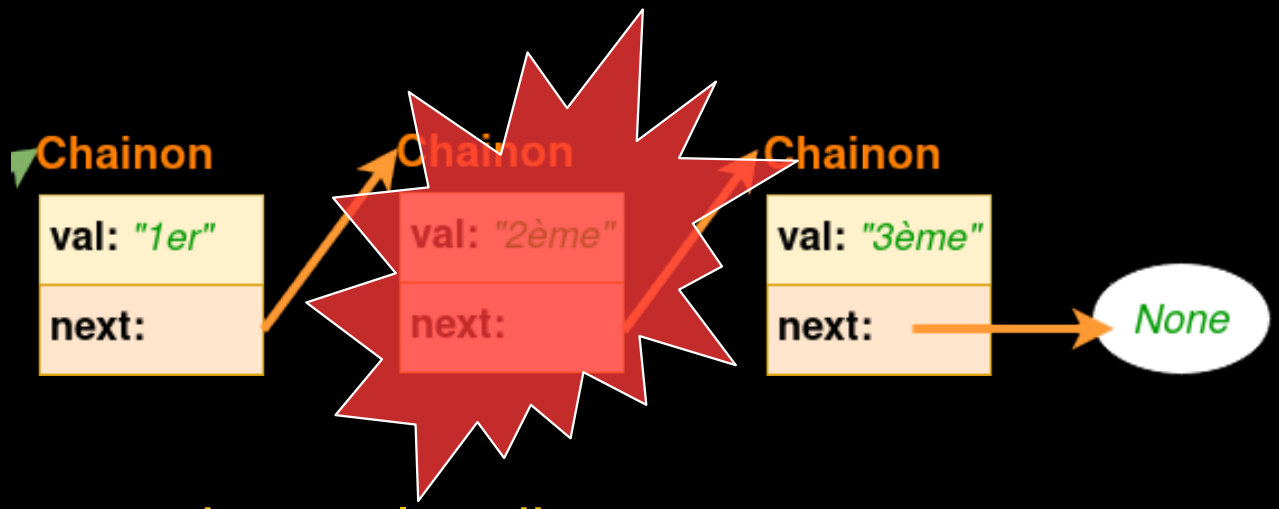
On ne peut pas supprimer le premier !

Supprimer un maillon : 4 cas

Cas terminal

Cas 3 :
Supprimer le maillon **suivant**

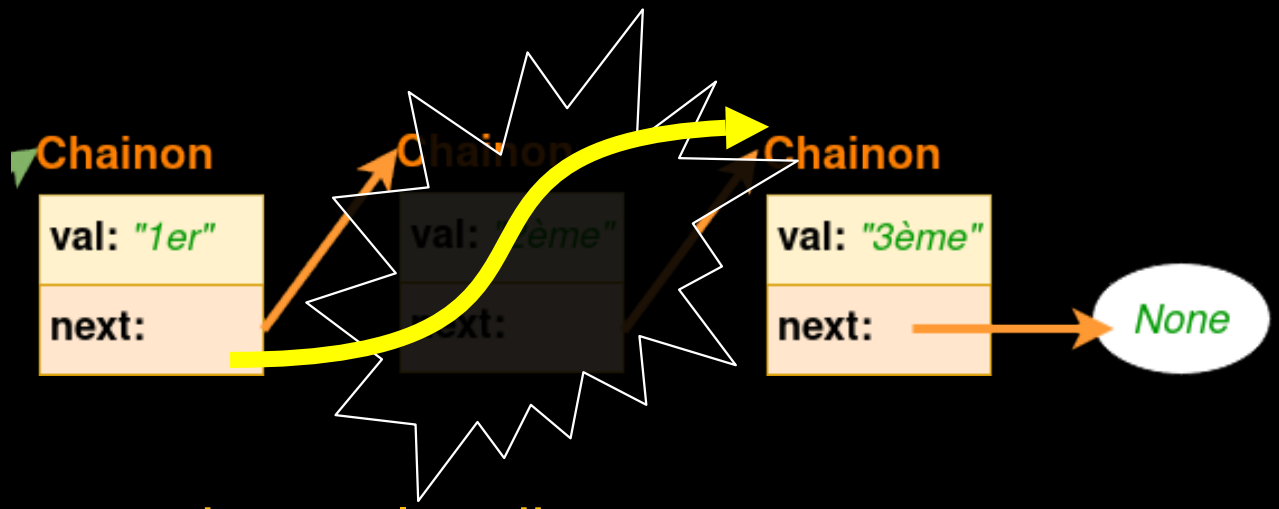
Supprimer le **suivant**



Facile ! On change l'attribut *next*

```
chaine = Chainon("1er", Chainon("2ème", Chainon("3ème")))
assert str(chaine) == "1er -> 2ème -> 3ème -> None"
chaine.supprimer(1)
assert str(chaine) == "1er -> 3ème -> None"
```

Supprimer le **suivant**



Facile ! On change l'attribut *next*

```python
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste chainée. """
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
    if indice == 0:
        self.val = self.next.val
        self.next = self.next.next
    elif indice == 1:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

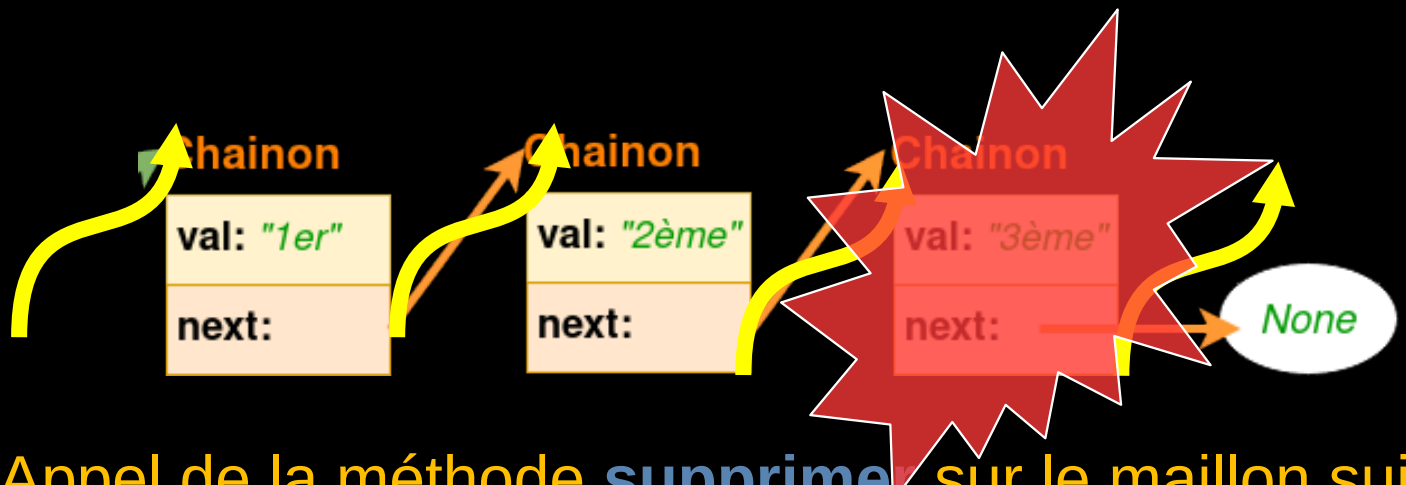
Facile ! On change l'attribut *next*

Supprimer un maillon : 4 cas

Parcours récursif

Cas 4 :
Il faut **avancer**
pour trouver
le maillon
à supprimer

Avancer



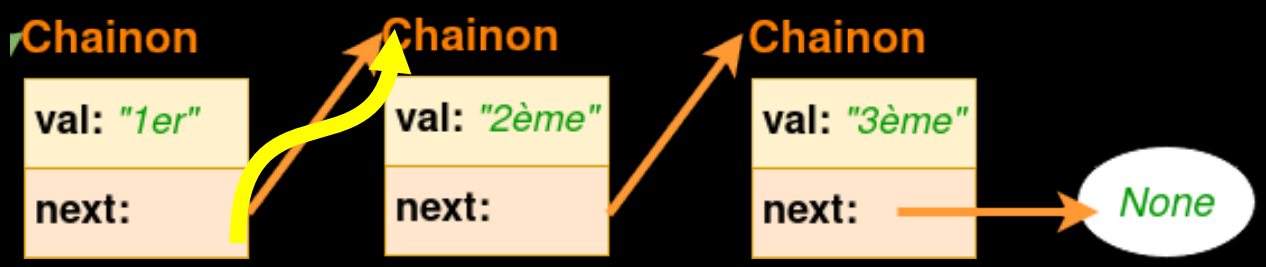
Appel de la méthode **supprimer** sur le maillon suivant

Parcours récursif

```
chaine = Chainon("1er", Chainon("2ème", Chainon("3ème")))
assert str(chaine) == "1er -> 2ème -> 3ème -> None"
chaine.supprimer(2)
assert str(chaine) == "1er -> 2ème -> None"
```

```
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste chainée. """
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
    if indice == 0:
        self.val = self.next.val
        self.next = self.next.next
    elif indice == 1:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

Appel de la méthode **supprimer** sur le maillon suivant

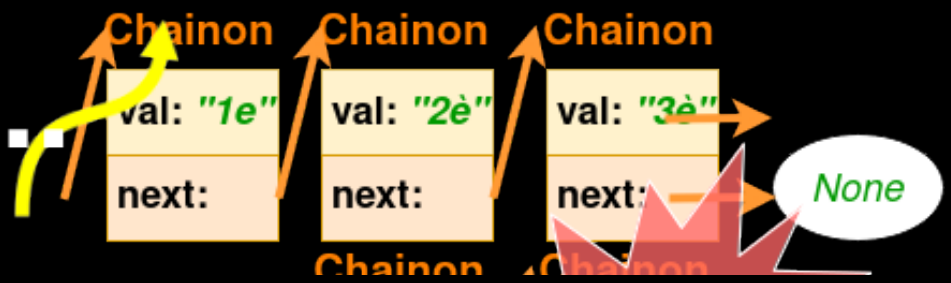


```
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste chainée. """
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
    if indice == 0:
        self.val = self.next.val
        self.next = self.next.next
    elif indice == 1:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

Appel de la méthode **supprimer** sur le maillon suivant

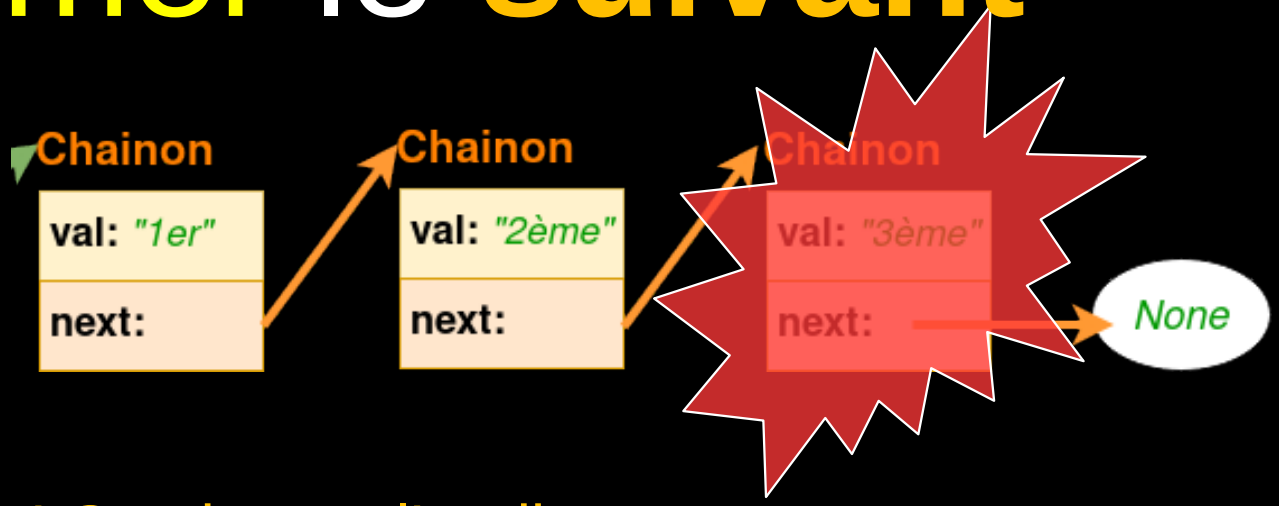
Parcours récursif

```
chaine.supprimer(2)
```



Retour au cas n°2 :

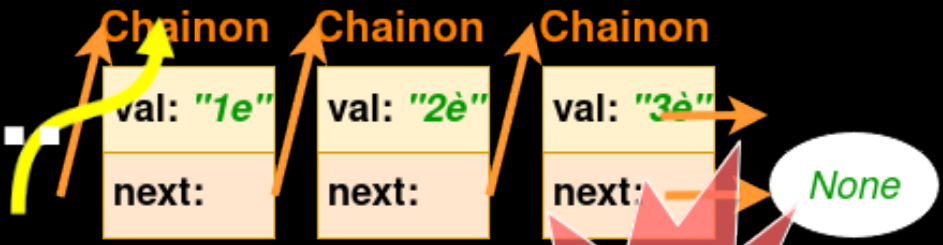
Supprimer le **suivant**



Facile ! On change l'attribut *next*

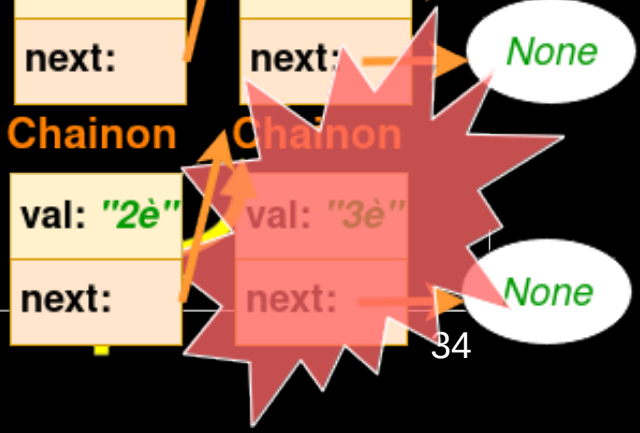
Parcours récursif

```
chaine.supprimer(2)
```



Cas terminal

```
chaine.supprimer(1)
```



```python
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste chainée. """
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
    if indice == 0:
        self.val = self.next.val
        self.next = self.next.next
    elif indice == 1:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

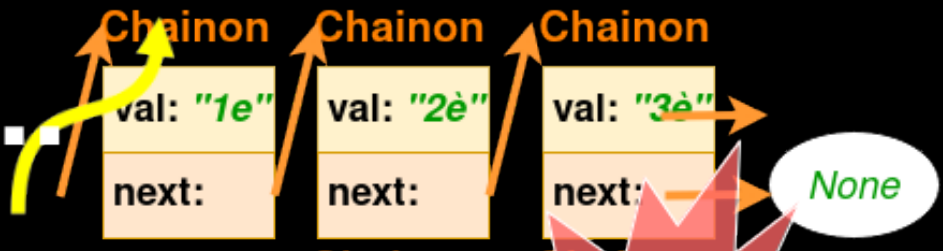
Facile ! On change l'attribut *next*

Supprimer le suivant

Le maillon suivant est supprimé.

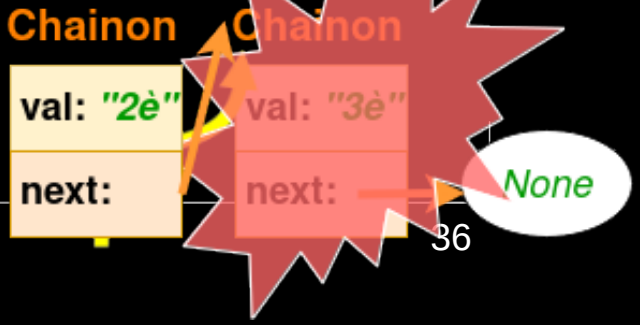
Parcours récursif

```
chaine.supprimer(2)
```



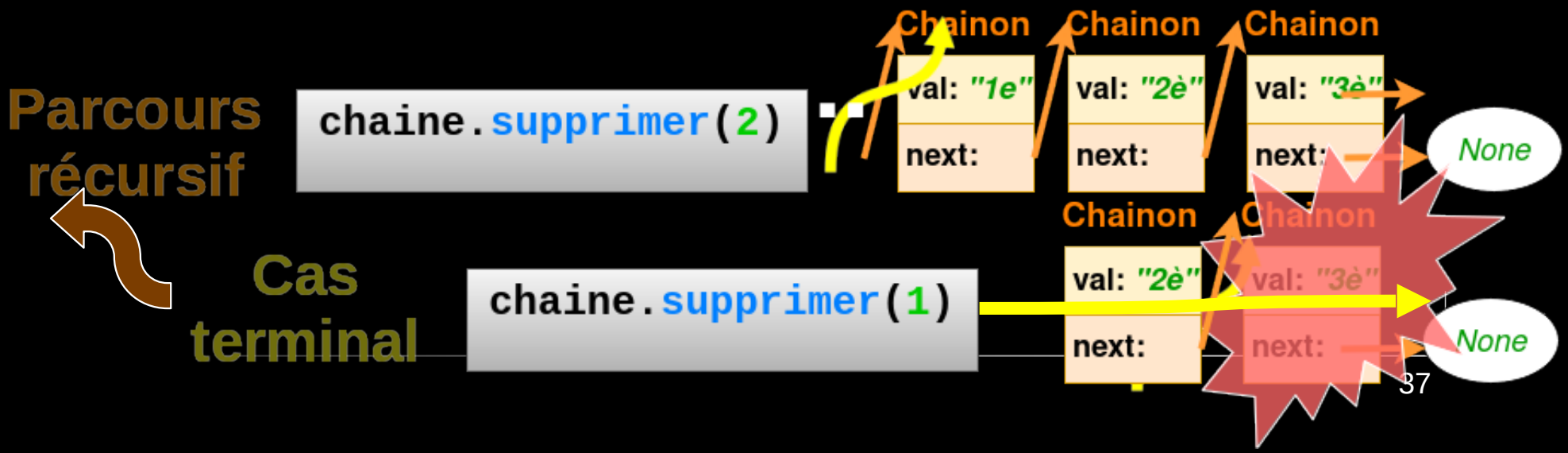
Cas terminal

```
chaine.supprimer(1)
```

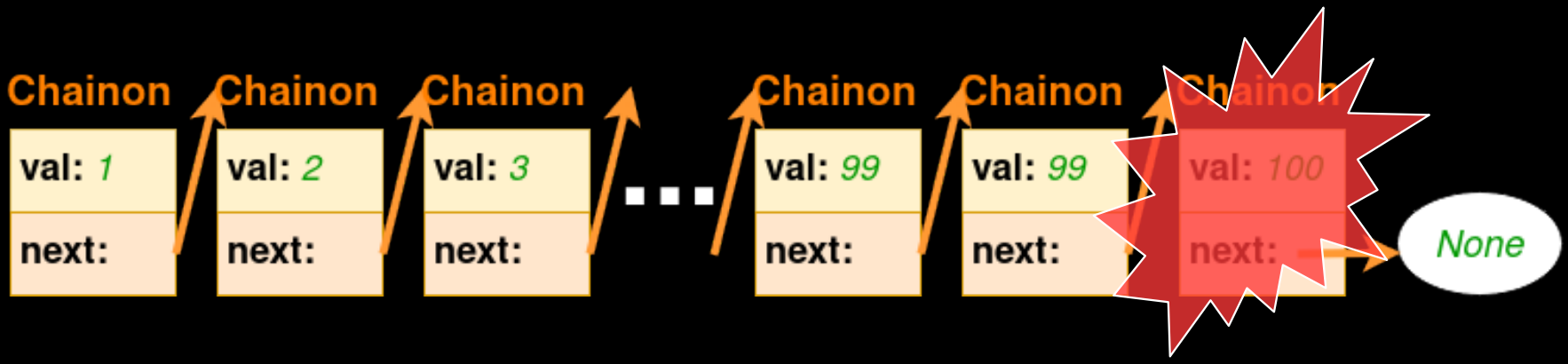


Supprimer le suivant

Fin de la pile d'appel ! Rapide et efficace :)



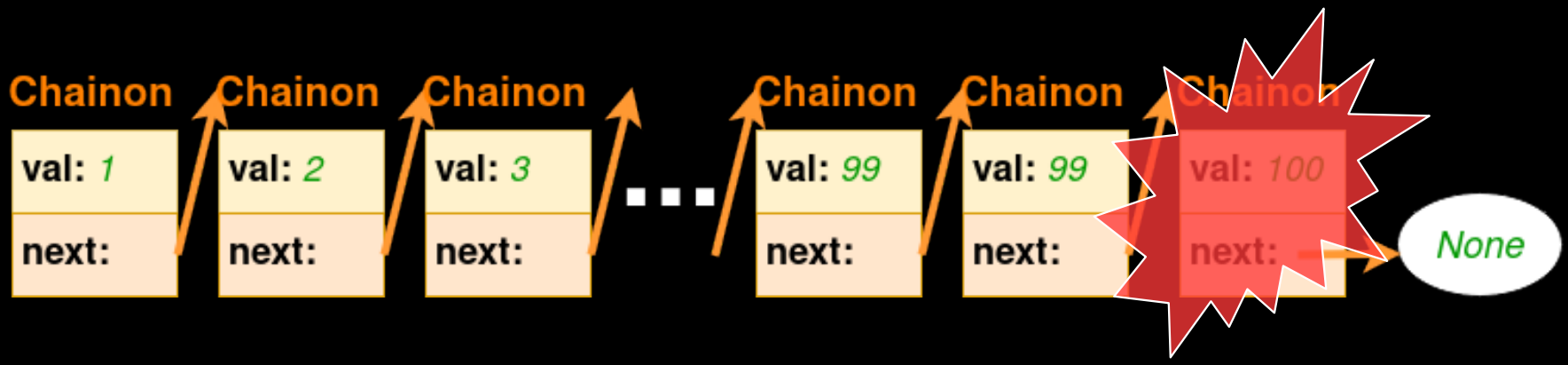
Supprimer le suivant



Et si on doit supprimer le dernier maillon d'une longue chaîne ?

```
chaine = Chainon(1, Chainon(2, Chainon(3 etc...)))
assert str(chaine) == "1 -> 2 -> ... -> 99 -> 100 -> None"
chaine.supprimer(99)
assert str(chaine) == "1 -> 2 -> ... -> 99 -> None"
```

Supprimer le **suivant**

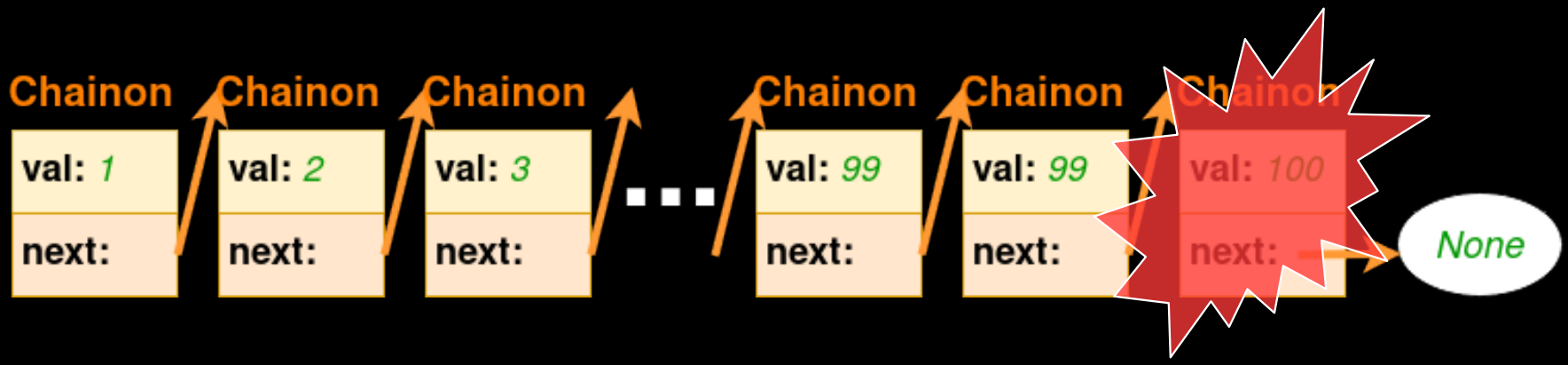


Et si on doit supprimer le dernier maillon d'une longue chaîne ?

```
def tester_grande_chaine(n):
    chaine = Chainon(0)
    for i in range(1, n):
        chaine.inserer(i, i)
    chaine.supprimer(n-1)

tester_grande_chaine(100)
```

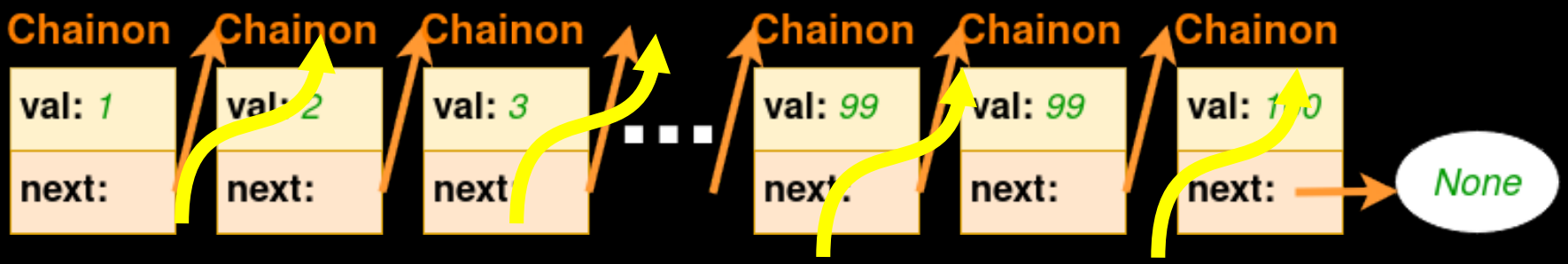
Supprimer le suivant



Et si on doit supprimer le dernier maillon d'une longue chaîne ?

On doit parcourir tous les maillons les un après les autres !

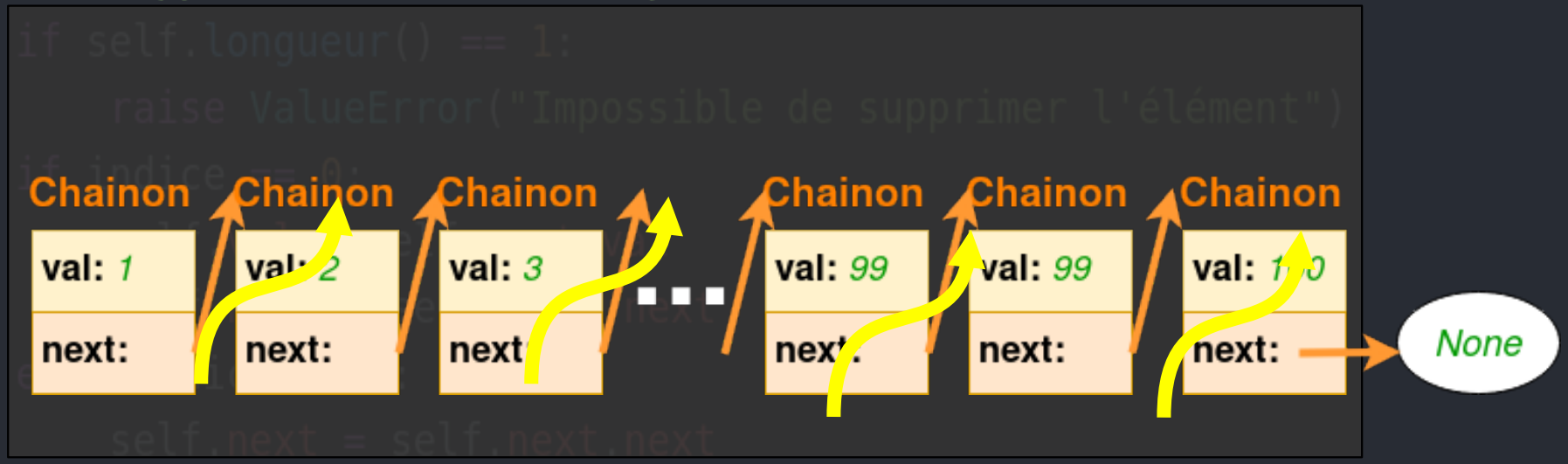
Avancer



Appel de la méthode **supprimer** sur le maillon suivant

Parcours récursif

```
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste chainée. """
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
    if indice == 0:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

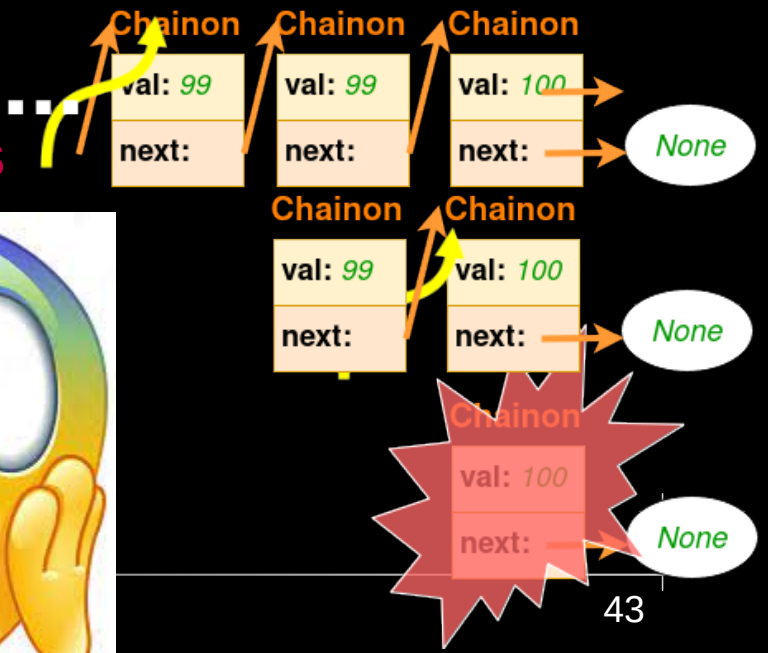


Appel de la méthode **supprimer** sur le maillon suivant

On doit parcourir tous les maillons les un après les autres

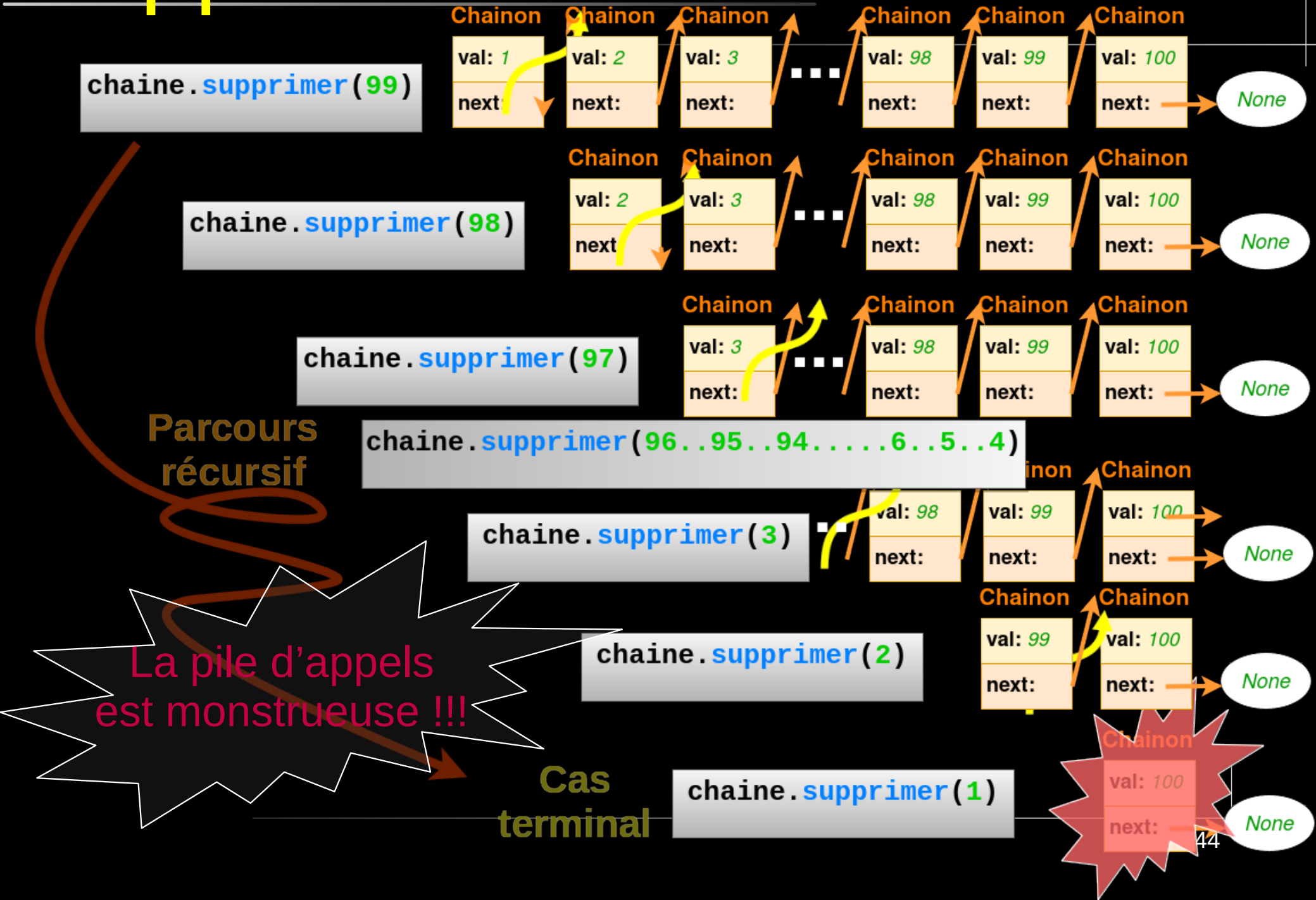
Supprimer le suivant

On doit parcourir tous les maillons les un après les autres



La pile d'appels est monstrueuse !!!

Supprimer le suivant



Supprimer le **suivant**

Pour une chaîne
avec **3 000** maillons :

```
def tester_grande_chaine(n):
    chaine = Chainon(0)
    for i in range(1, n):
        chaine.inserer(i, i)
    chaine.supprimer(n-1)

tester_grande_chaine(3000)
```

Supprimer le **suivant**

Pour une chaîne
avec **3 000** maillons :

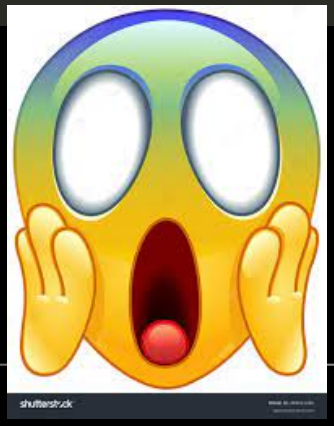
```
def tester_grande_chaine(n):
    chaine = Chainon(0)
    for i in range(1, n):
        chaine.inserer(i, i)
    chaine.supprimer(n-1)

tester_grande_chaine(3000)
```

```
RecursionError:

maximum recursion depth exceeded
```

La pile d'appels
est monstrueuse !!!



Supprimer un maillon

```
def supprimer(self:C, indice:int) ->None:
    """ Supprime le chainon en position indice de la liste cha
    if self.longueur() == 1:
        raise ValueError("Impossible de supprimer l'élément")
    if indice == 0:
        self.val = self.next.val
        self.next = self.next.next
    elif indice == 1:
        self.next = self.next.next
    else:
        self.next.supprimer(indice-1)
```

Cas 1

Cas 2

Cas 3

Cas 4

Conclusion

L'implantation récursive des listes chaînées :

- Utilisation de la mémoire : optimale
- Temps d'exécution :
  - Rapide si on supprimer le 1er élément.
  - Monstrueux si on supprime le dernier !
- Code source : concis et complexe