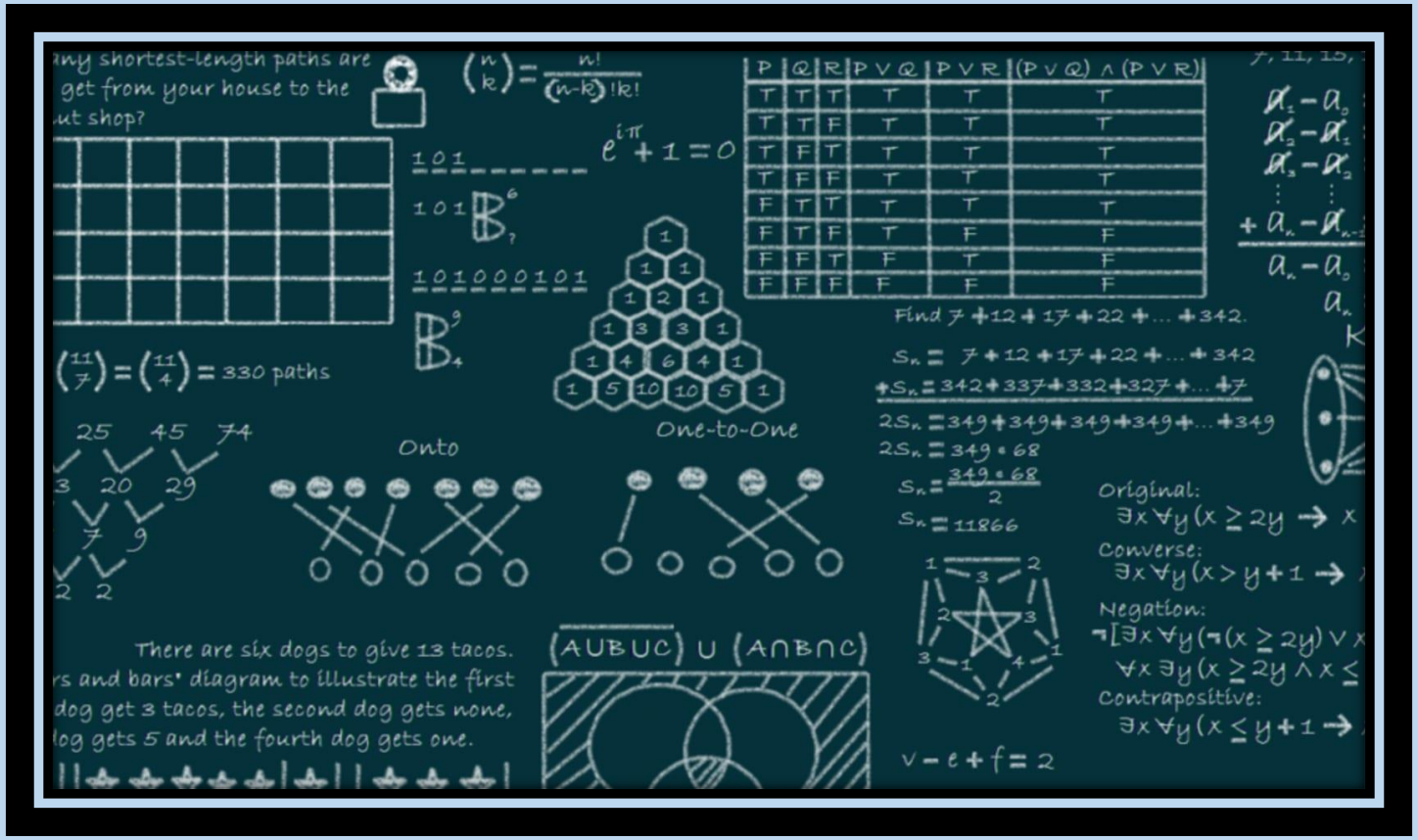


FÖRELÄSNING 2

Datastrukturer och algoritmer
KYH – 2022 HT

Andreas Nilsson Ström

Om Omniway är nere:

- Ni kan nå mig via:
  - andreas.nilssonstrom@kyh.se
  - andreas.nilsson@natrolit.se

Agenda

- Repetition av gårdagens höjdpunkter
- Mer om länkade listor!
- Övningar och labbande

Repetition

Länkade listor forts.

Igår lärde vi oss

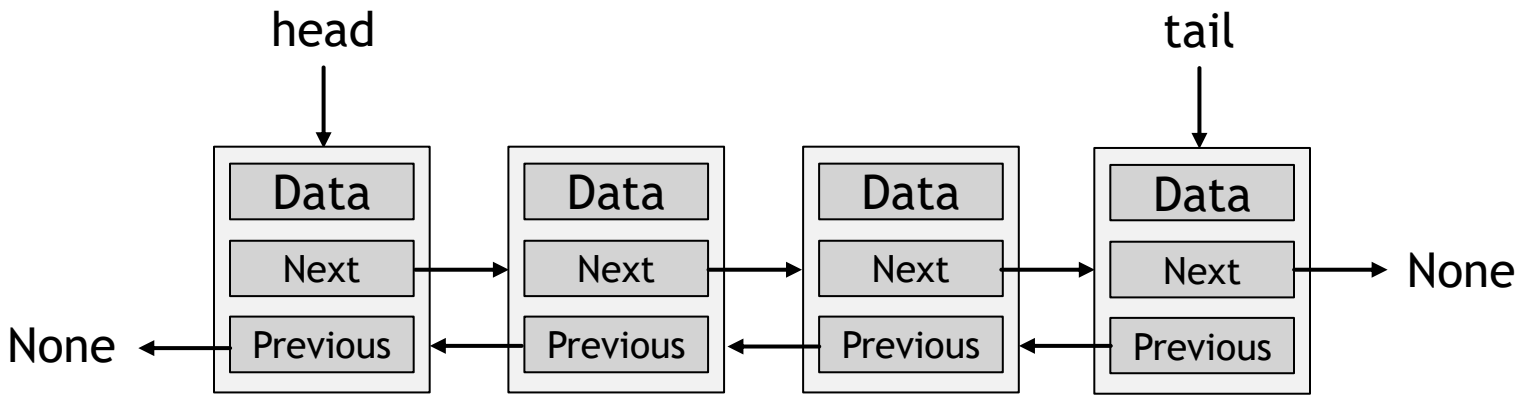
- Att skapa en Node, som håller data och en pekare till nästa nod
- Att lägga till data på slutet av vår länkade list (append)
- Räkna antalet noder, summera alla värden

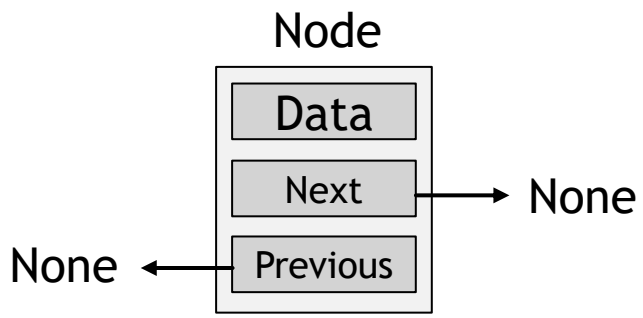
Dagens agenda för länkade listor

- Varianter av länkade listor
- Lägga till noder på andra platser i listan
- Ta bort noder från olika platser
- Extra: Göra koden mer Pythonisk? (Och vad betyder "Pythonisk"?)

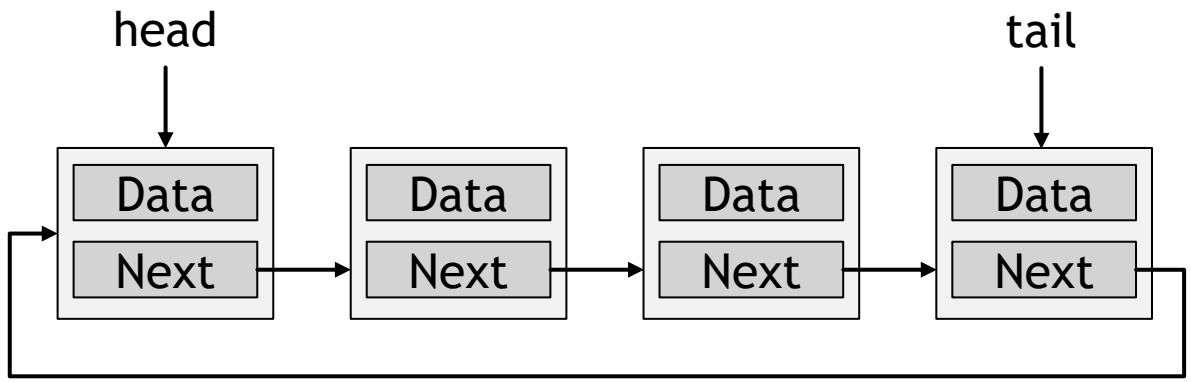
Flera varianter av länkade listor

Dubbel-länkade listor: Next och Previous



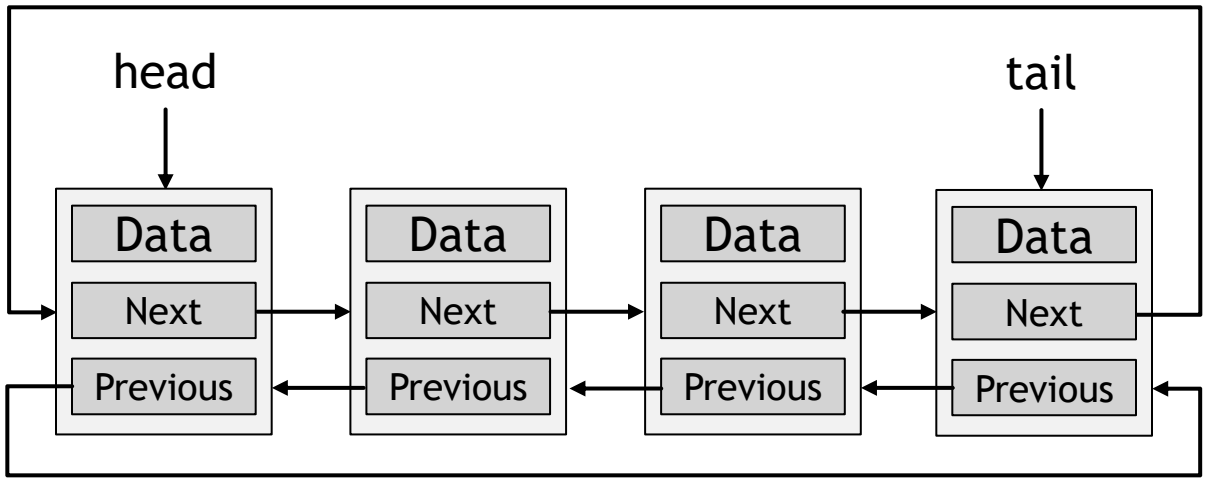


Cirkulära listor



Baserat på enkel-länkad lista

Cirkulära listor



Baserat på dubbel-länkad lista

Garbage collection

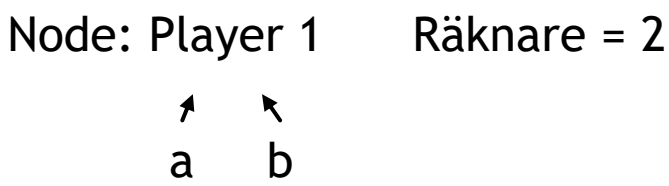
Garbage collection

- RAM-minne är inte oändligt
- Vi ska bara använda så mycket vi behöver
- Hur frigör vi minne när vi inte behöver det längre?

Garbage collection

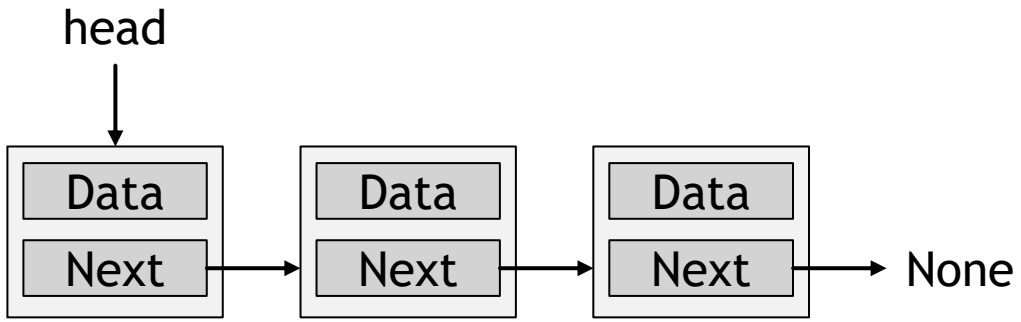
- När vi skapar ett objekt så håller Python koll på hur många variabler som pekar på objektet
- Om vi skapar en variabel: Räknaren för det objektet = 1
- Kopiera variabeln: räknare + 1

```
def example():
    a = Node("Player 1")
    b = a
```



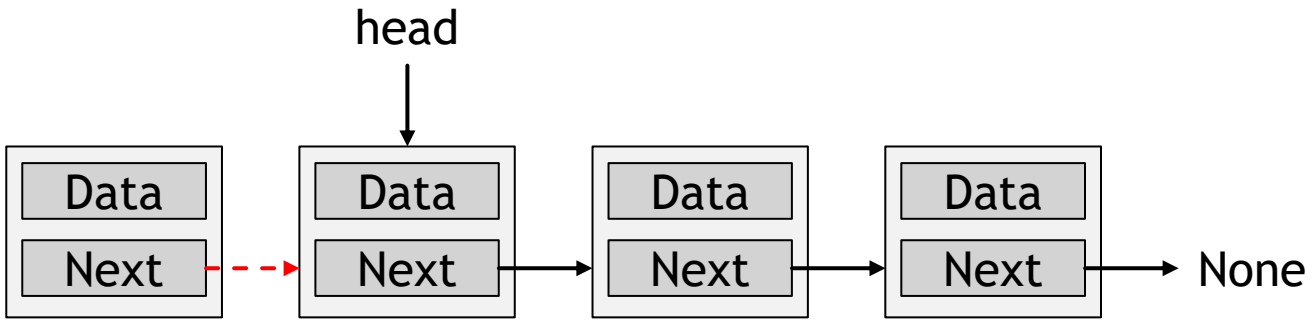
Om räknaren går ned till 0: Ta bort objektet ur minnet
När vi lämnar example() så finns inte a och b längre.
Då tar Python bort listan ur minnet.

Lägg till i början – prepend()



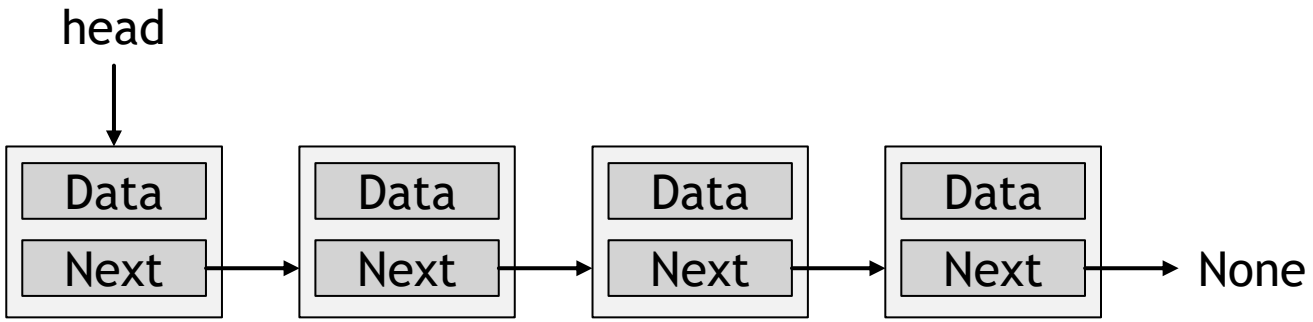

```
def prepend(self, data):
    # ...
```

Lägg till i början – prepend()



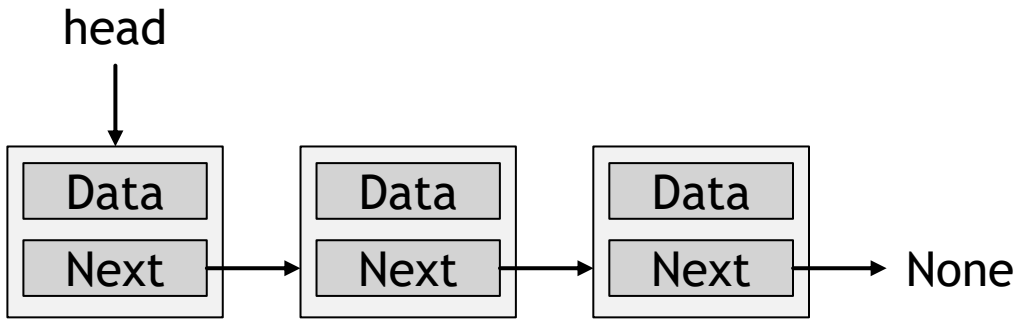

```
def prepend(self, data):
    # 1. Skapa en ny nod
    # 2. Sätt den nya nodens "next"
    #    till self.head
```

Lägg till i början – prepend()



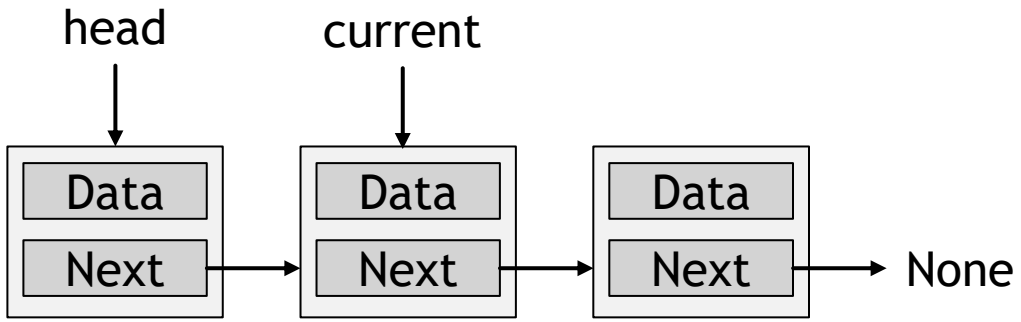

```
def prepend(self, data):
    # 1. Skapa en ny nod
    # 2. Sätt den nya nodens "next"
    #    till self.head
    # 3. Flytta self.head till den
         nya noden
```

Lägg till i mitten – insert()



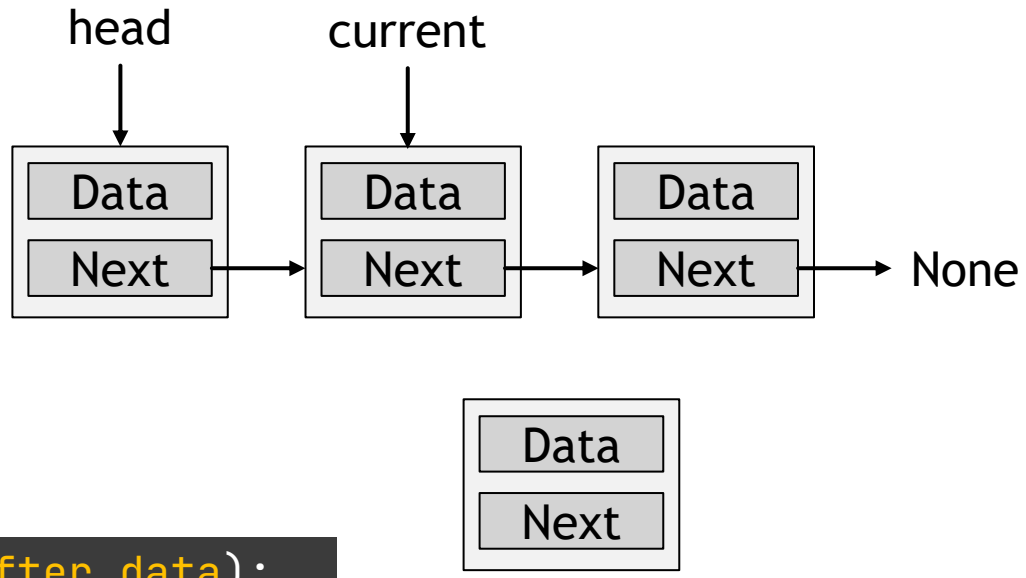

```
def insert(self, data, after_data):
    # ...
```

Lägg till i mitten – insert()



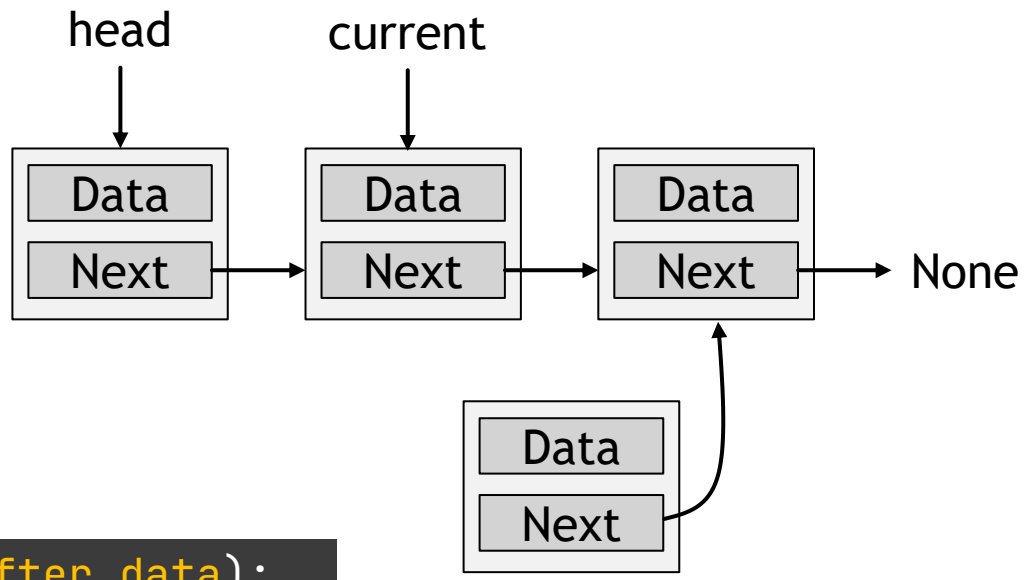

```
def insert(self, data, after_data):
    # 1. Hitta rätt ställe i listan
```

Lägg till i mitten – insert()



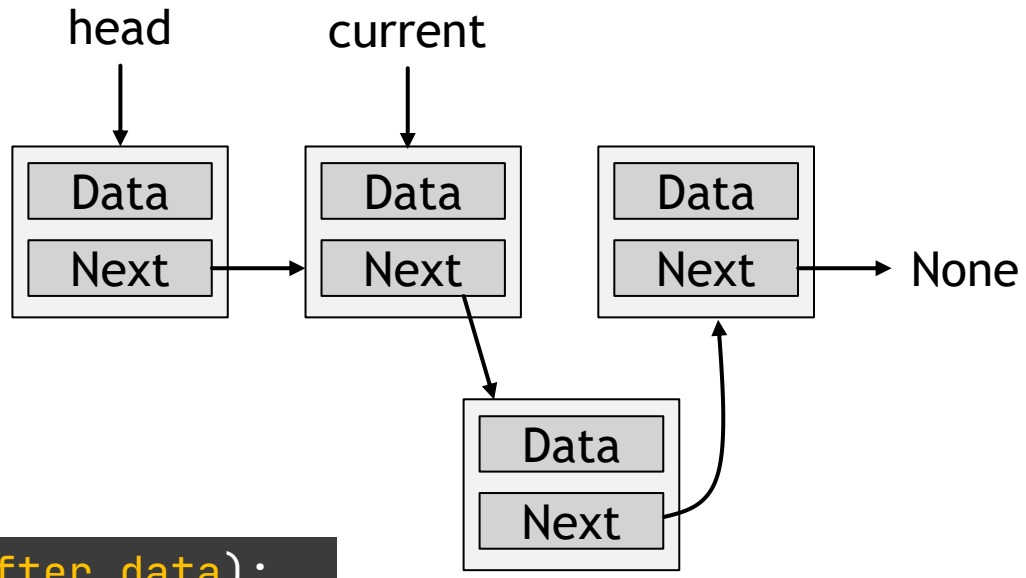

```
def insert(self, data, after_data):
    # 1. Hitta rätt ställe i listan
    # 2. Skapa en ny nod
```

Lägg till i mitten – insert()



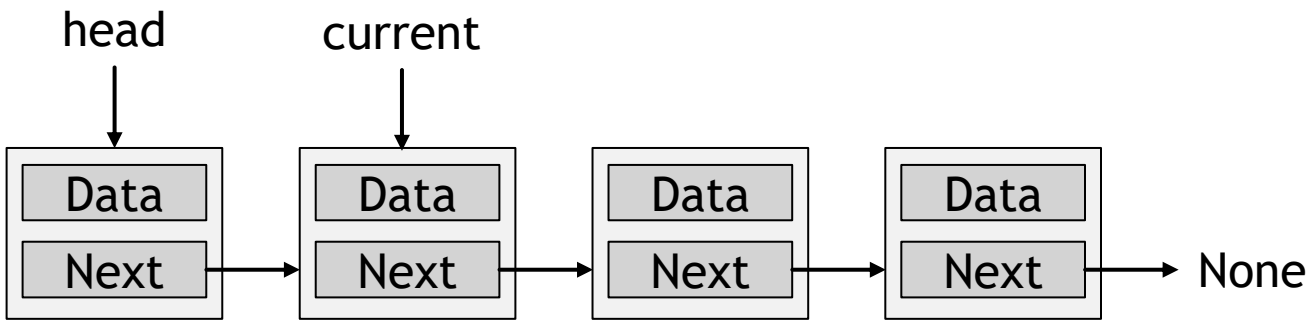

```
def insert(self, data, after_data):
    # 1. Hitta rätt ställe i listan
    # 2. Skapa en ny nod
    # 3. Sätt den nya nodens "next"
    #    till nästa nod
```

Lägg till i mitten – insert()



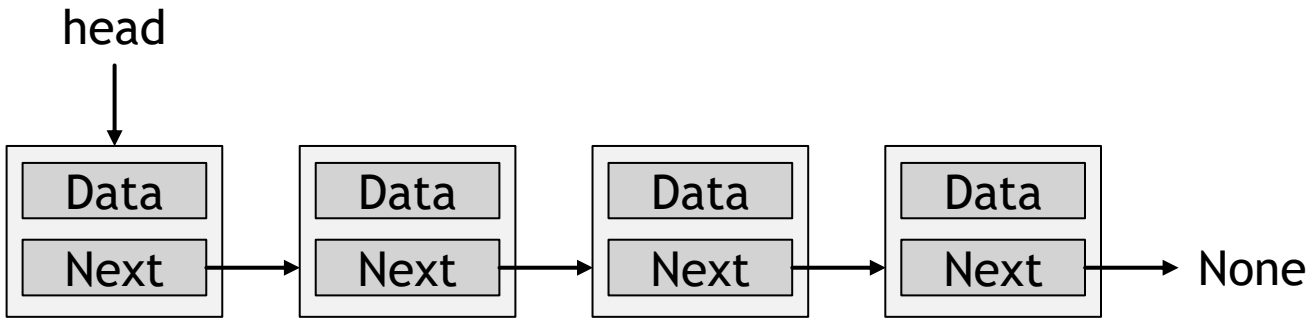

```
def insert(self, data, after_data):
    # 1. Hitta rätt ställe i listan
    # 2. Skapa en ny nod
    # 3. Sätt den nya nodens "next"
    #    till nästa nod
    # 4. Sätt nuvarande nodens
         "next" till nya noden
```

Lägg till i mitten – insert()



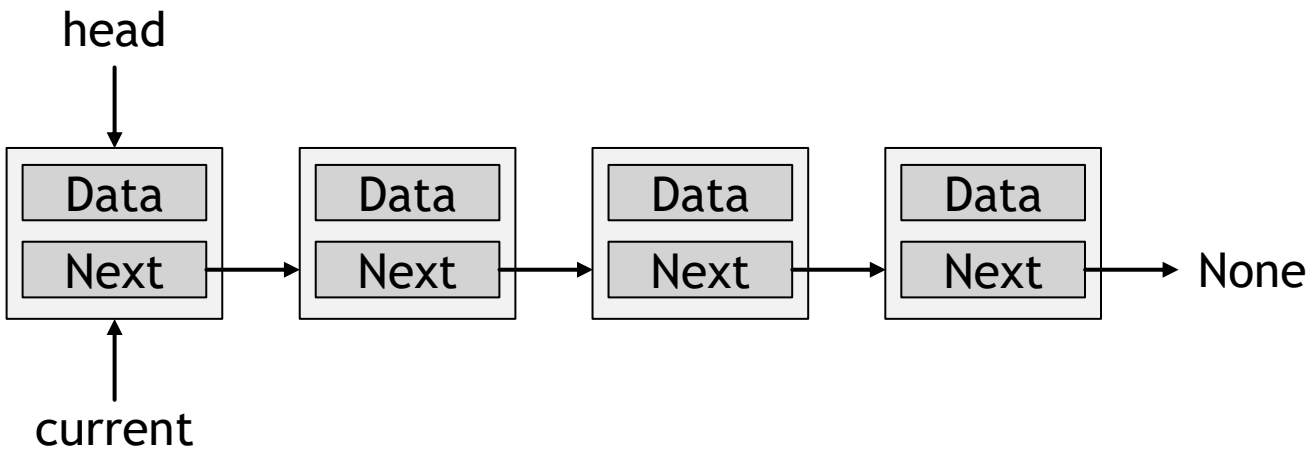

```
def insert(self, data, after_data):
    # 1. Hitta rätt ställe i listan
    # 2. Skapa en ny nod
    # 3. Sätt den nya nodens "next"
    #    till nästa nod
    # 4. Sätt nuvarande nodens
         "next" till nya noden
```

Ta bort i början



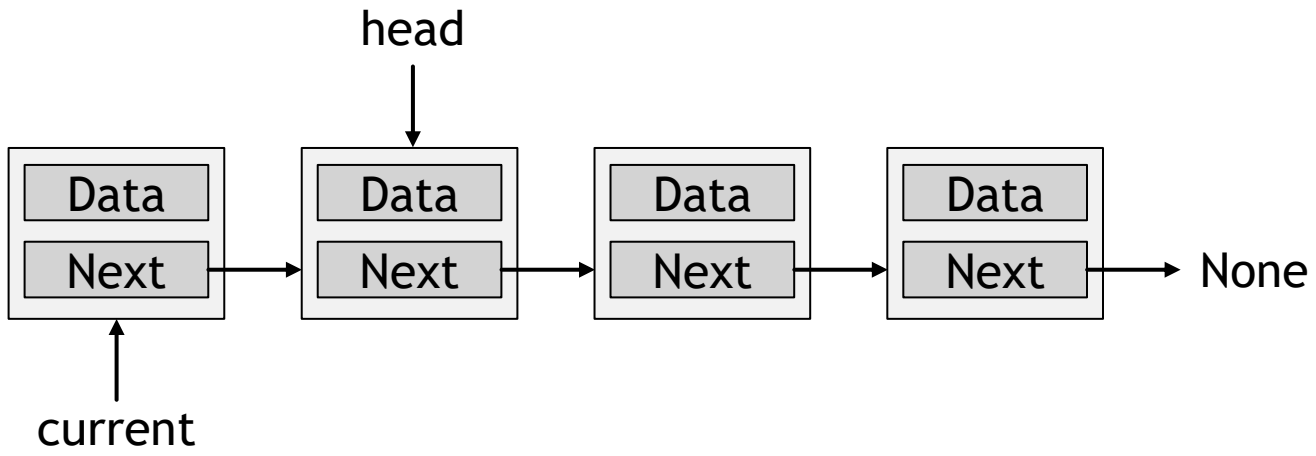

```
def remove_first_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
```

Ta bort i början



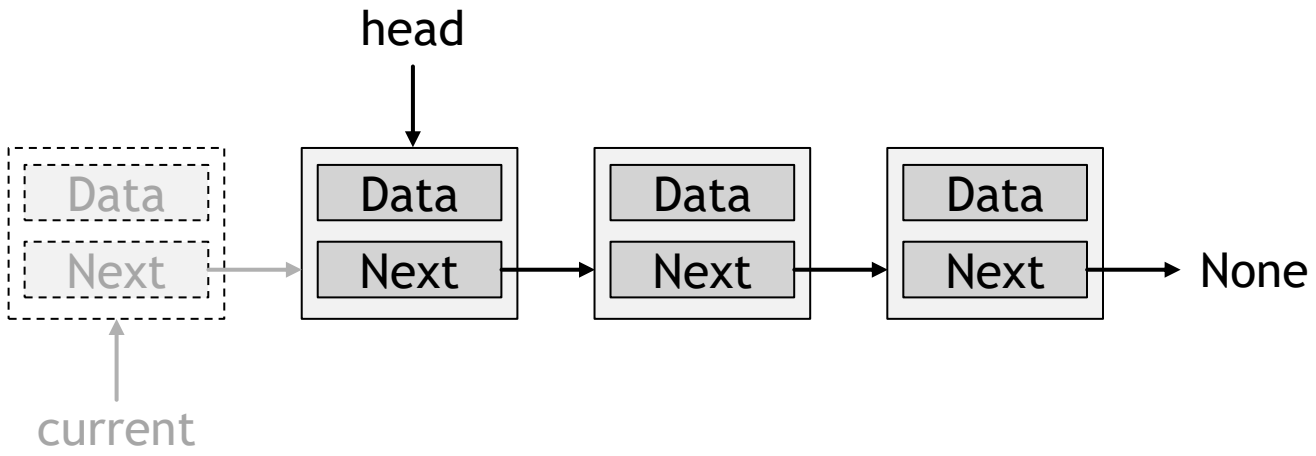

```
def remove_first_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa en tillfällig pekare
         "current"
```

Ta bort i början



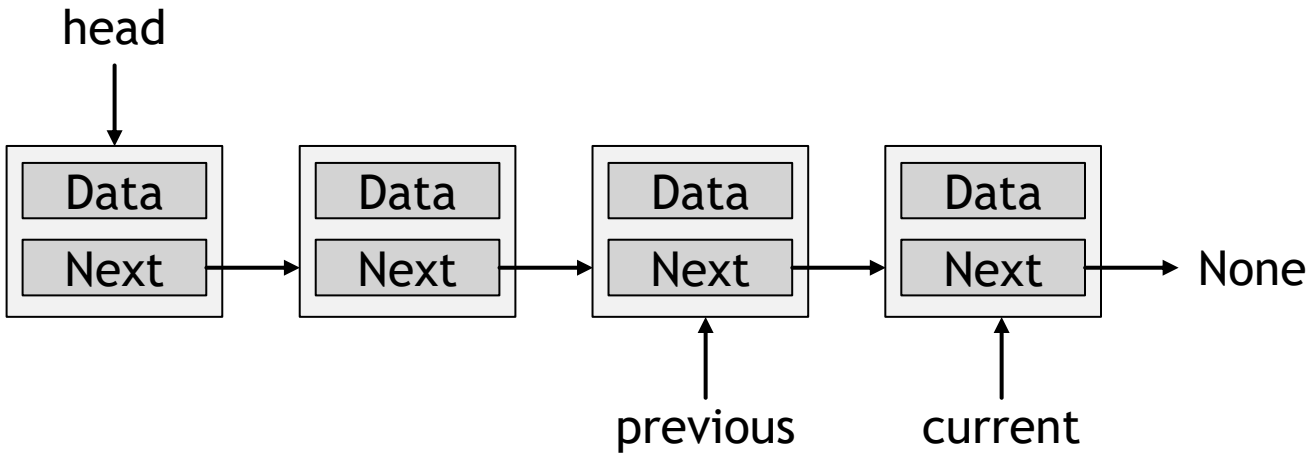

```
def remove_first_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa en tillfällig pekare
         "current"
    # 3. Flytta "self.head" till
         nästa nod
```

Ta bort i början



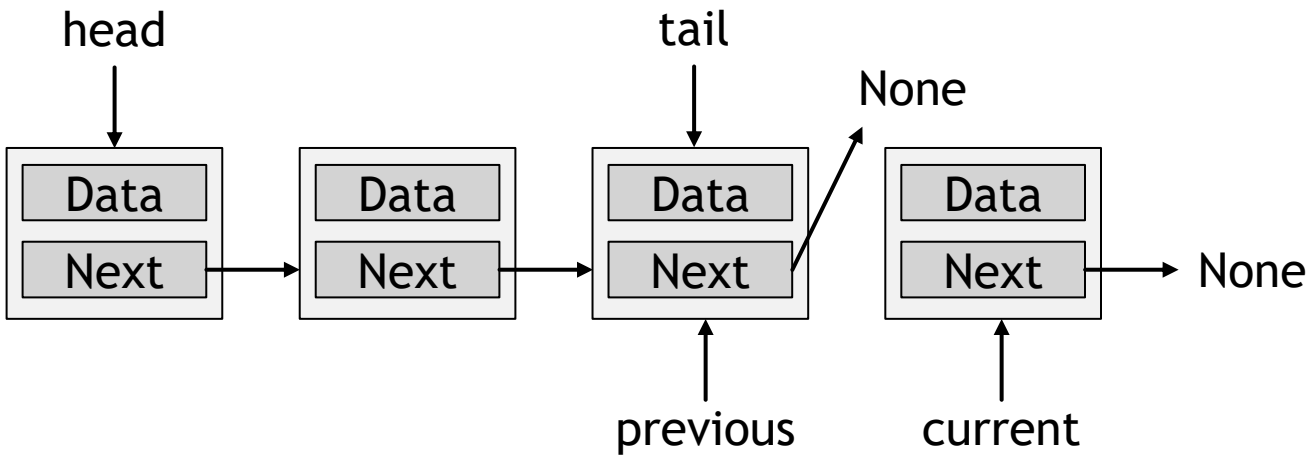

```
def remove_first_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa en tillfällig pekare
         "current"
    # 3. Flytta "self.head" till
         nästa nod
```

Ta bort i slutet



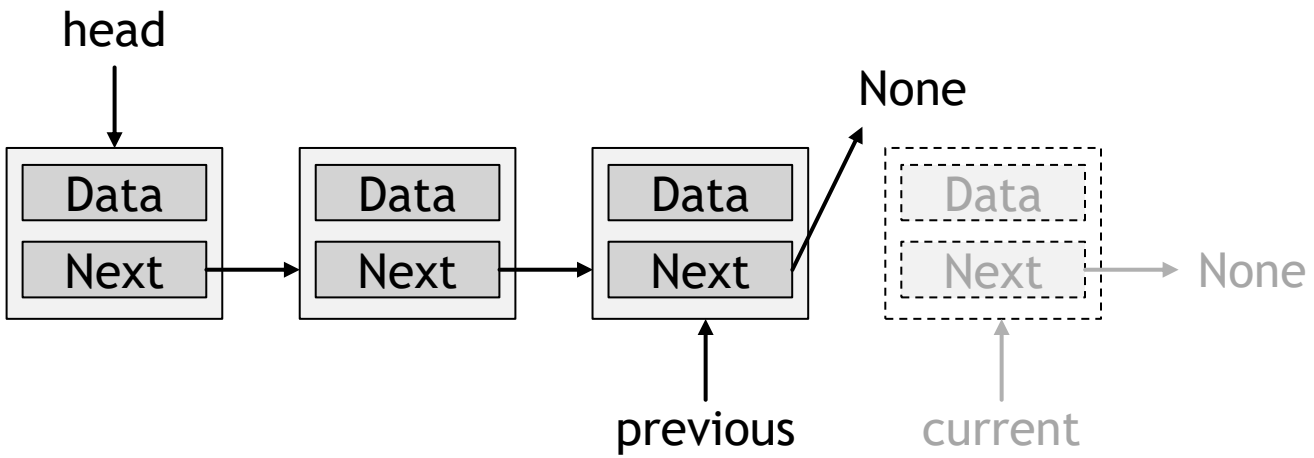

```
def remove_last_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         sista noden:
         (current.next är None)
```

Ta bort i slutet



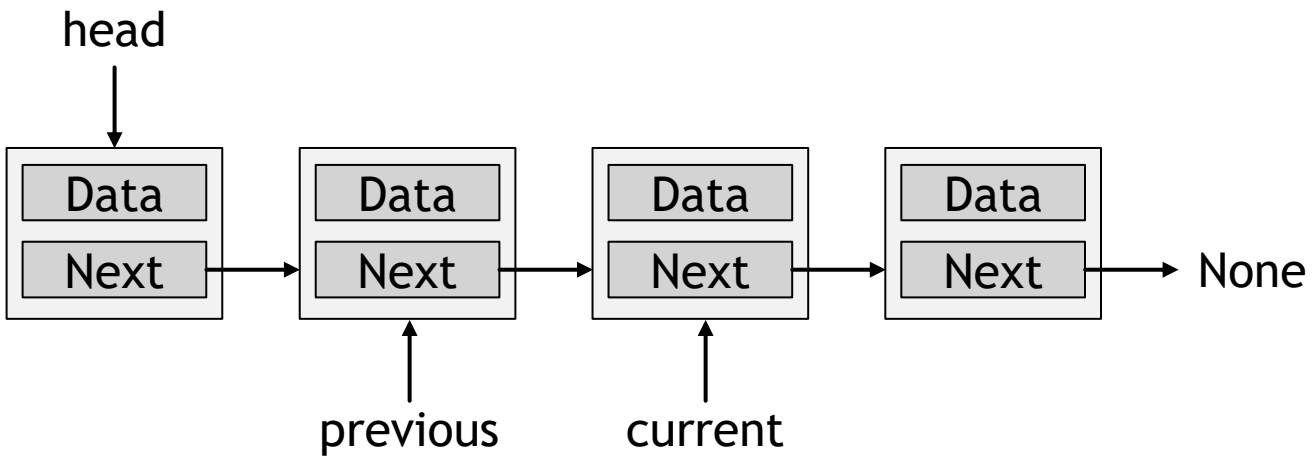

```
def remove_last_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         sista noden
         (current.next är None)
    # 4. Sätt previous.next = None
```

Ta bort i slutet



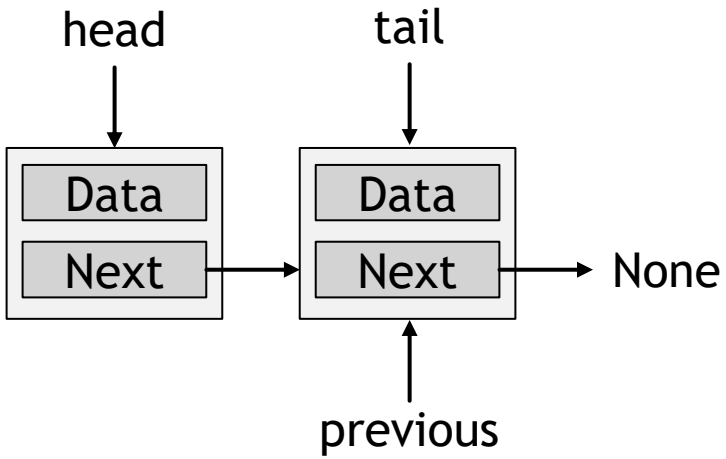

```
def remove_last_node(self):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         sista noden
         (current.next är None)
    # 4. Sätt previous.next = None
```

Ta bort i mitten



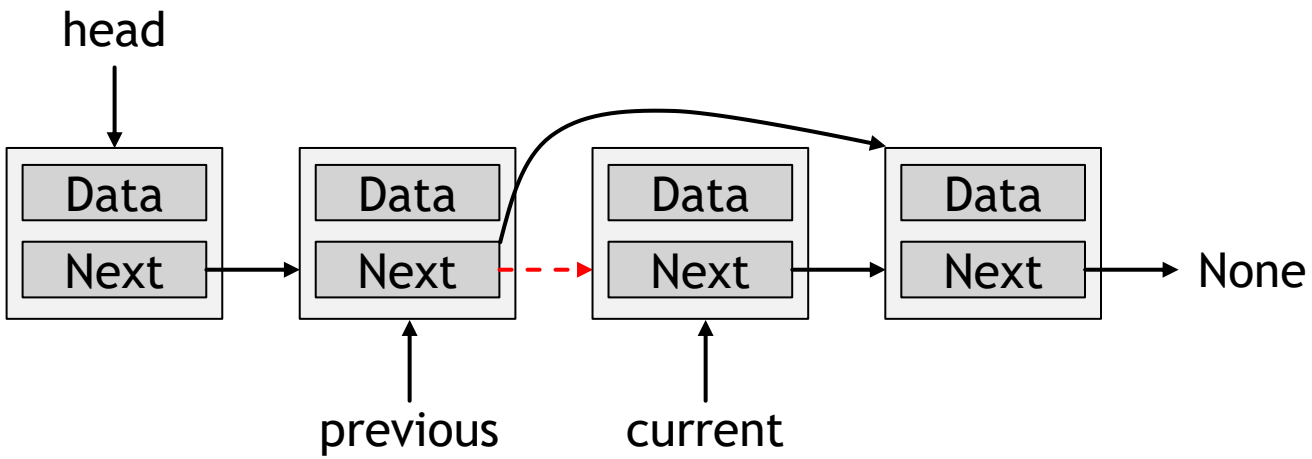

```
def remove_node(self, data):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         noden som ska tas bort
```

Ta bort i mitten



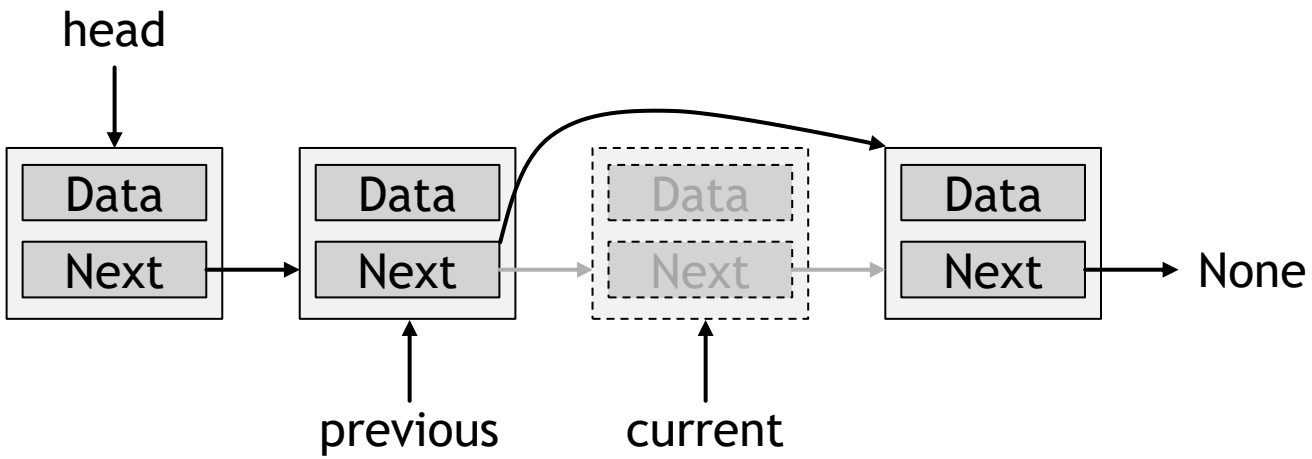

```
def remove_node(self, data):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         noden som ska tas bort
```

Ta bort i mitten




```
def remove_node(self, data):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         noden som ska tas bort
    # 4. Sätt previous.next = current.next
         (Om current == head, sätt head.next)
```

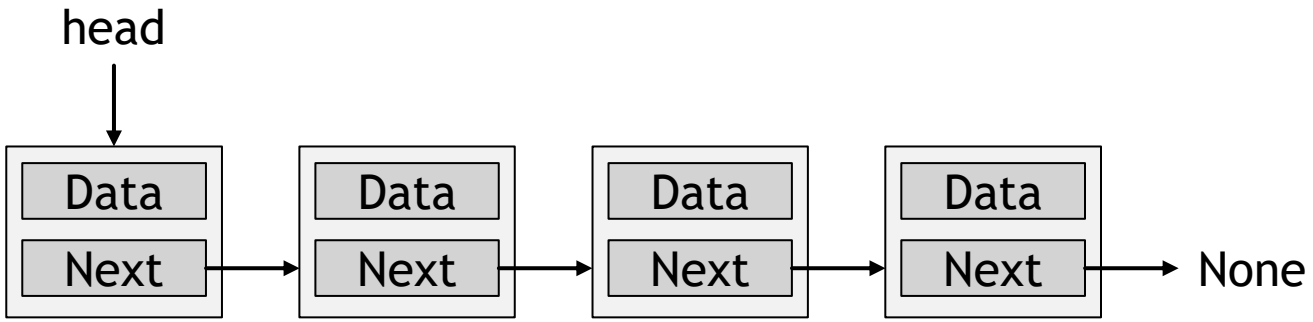
Ta bort i mitten




```
def remove_node(self, data):
    # (1. Om listan är tom:
          Ge IndexError)
    # 2. Skapa två tillfälliga pekare
         "current" och "previous"
    # 3. Stega tills "current" är
         noden som ska tas bort
    # 4. Sätt previous.next = current.next
         (Om current == head, sätt head.next)
```

Facit

Lägg till i början – prepend()




```
def prepend(self, data):
    new_node = Node(data)
    new_node.next = self.head
    self.head = new_node
```

Extra: Vad betyder "Pythonisk"?

- Koden vi skriver ska fungera på liknande sätt som etablerade standarder
- "Python ska se ut som Python för andra Python-programmerare"

- Vad gör vi som bryter mot detta?
  - Till exempel: count()

    llist.count() är inte så man skriver i alla andra typer av listor
    Hur använder vi len(llist) istället?

Extra: Pythoniska detaljer

- Längd med len(linked_list):
  - Implementera __len__()
- Finns värdet i listan? ("a" in linked_list?)
  - Implementera __contains__()
- Loopa igenom vår lista med en for-each-loop:
  - Implementera __iter__()

__iter__(): Vad gör yield?

- "yield" är nästan samma sak som "return"
  - Metoder som använder "yield" kallas "Generatorer" istället för "Metoder"

- Istället för att avsluta metoden och skicka tillbaka ett värde:
  - Pausa och kom ihåg platsen, skicka tillbaka ett värde
  - Om vi anropar funktionen igen: Återuppta efter där vi gjorde "yield"

- Dock: Inte riktigt som en vanlig metoden, utan vi måste loopa över metoden
  - Anropa metoden direkt ger inte något särskilt användbart

```
def test_yield():
    yield 1
    yield 2
    yield 3

>>> test_yield()
  <generator object b at 0x...>
```

```
for num in test_yield():
    print(num)

 1
 2
 3
```

__iter__(): Vad gör yield?

- Vad händer om vi har en oändlig generator?

```
def infinite_yield():
    n = 0
    while True:
        n += 1
        yield n
```

- For-loop blir lite jobbigt… Det blir en oändlig loop.
  - Men vi kan t.ex. anropa next() för att få ett värde i taget

```
>>> my_gen = infinite_yield()
>>> next(my_gen)
 1
>>> next(my_gen)
 2
>>> next(my_gen)
 3
```

Nästa gång…

- Flera nya varianter och termer! (Det sista om länkade listor! … ?)
  - Lite mer praktiskt, vad kan vi göra med dessa saker
  - Översikt över varianter och detaljer
- Vilka sätt har vi att jämföra datastrukturers körtid?
  - Hur pratar vi om det?
- Översikt över nästa vecka

Till nästa gång…

- Övningar: Hackerrank
  - https://www.hackerrank.com/domains/data-structures?filters%5Bsubdomains%5D%5B%5D=linked-lists
  - Till och med "Compare two linked lists"
- Mer läsning:
  - https://www.happycoders.eu/algorithms/array-vs-linked-list/