

Solutions

Each answer shows the steps, then the **result**.

2.1

- **14**
- **None**: 14 is the last node

2.2

- `head.next = head.next.next`

2.3

- it grows and shrinks at run time, with no fixed size
- inserting or deleting needs only pointer changes, no shifting

2.4

- it is defined by its operations, enqueue and dequeue, and its FIFO rule
- not by how it is stored: it can be an array or a linked list

3.1 C

- **None** marks the end

3.2 B

- an array computes the address directly; a list follows 500 pointers

3.3 B

- after the first statement `prev.next` is the new node, so it points to itself

3.4

(a) One answer:

```
def length(head):
    count = 0
    current = head
    while current is not None:
        count = count + 1
        current = current.next
    return count
```

- starts at `head` with a count of 0
- loops until `None`, moving with `current = current.next`
- counts each node and returns the count

(b)

- `head = head.next`

(c)

- `new.next = p.next`
- then `p.next = new`

(d)

- without `tail`, reaching the last node means traversing every node: $O(n)$
- with `tail`, set `tail.next = new` and `tail = new`: $O(1)$