APS106

UNIVERSITY OF TORONTO

linked lists and binary trees.

Week 12 | Lecture 1 (12.1)

if nothing else, write #cleancode

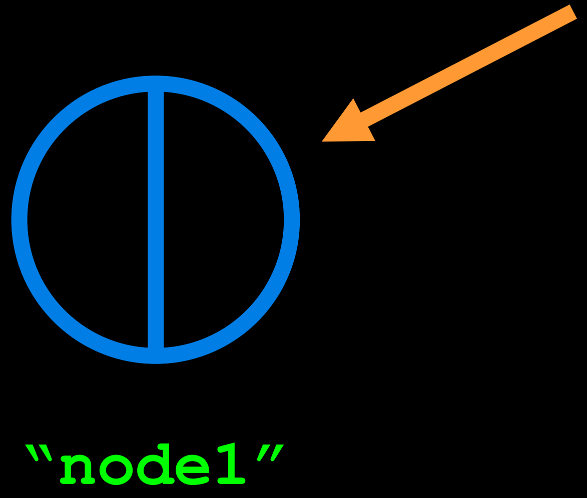
This Week's Content

- **Lecture 12.1**
  - **Linked lists, binary trees**
  - **Reading: Chapter 14**
- **Lecture 12.2**
  - Binary search trees
  - Reading: Chapter 14
- **Lecture 12.3**
  - Design Problem: 20 Questions

linked lists.

Linked Lists

- What is a linked list?

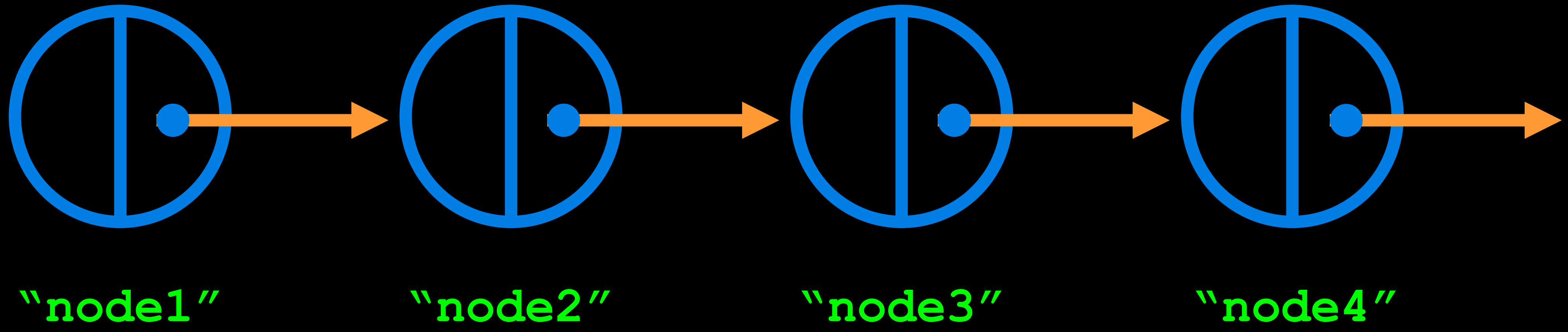


Check out this node.

Linked Lists

- What is a linked list?

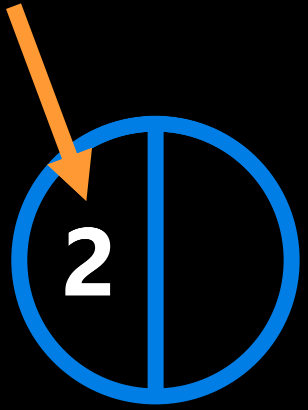
Connect a bunch of these together and we have a linked list.



Linked Lists

- What is a linked list?

Node value.



“node1”

A node can contain a value like a number.

The node value is stored in the `.cargo` attribute.

```
node1.cargo = 2
```

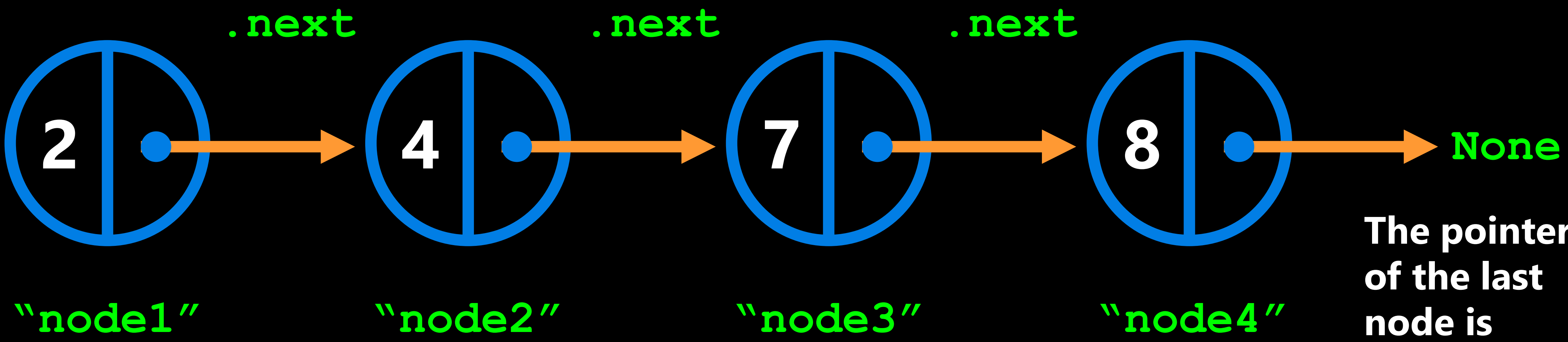
Linked Lists

- What is a linked list?

Value
Pointer

Pointers are used to connect each node to the next node in the list.

We can access a pointer using the `.next` attribute.



The pointer of the last node is `None`.

The Node Class

- Let's quickly revisit the Node class from last week.

Open your notebook

Click Link:
1. Node Class

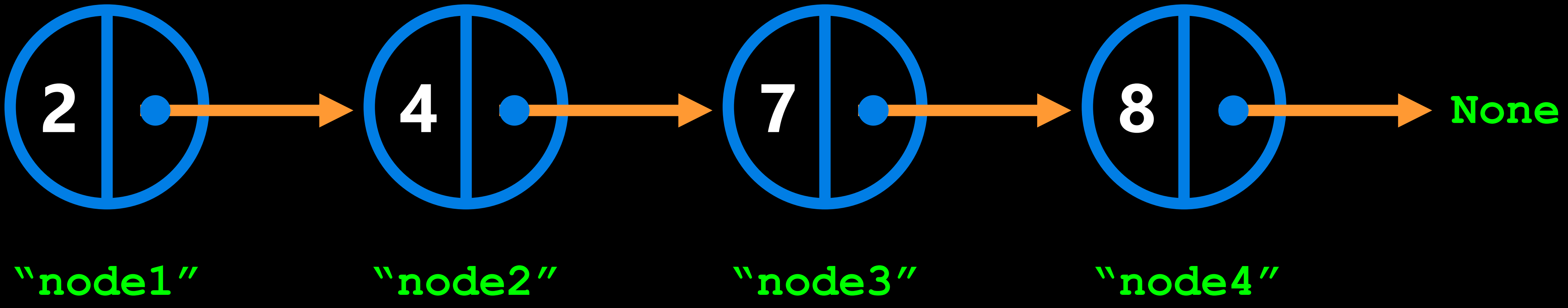
Linked Lists

- What is a linked list?

```
>>> node1.cargo
?
```

Value

Pointer

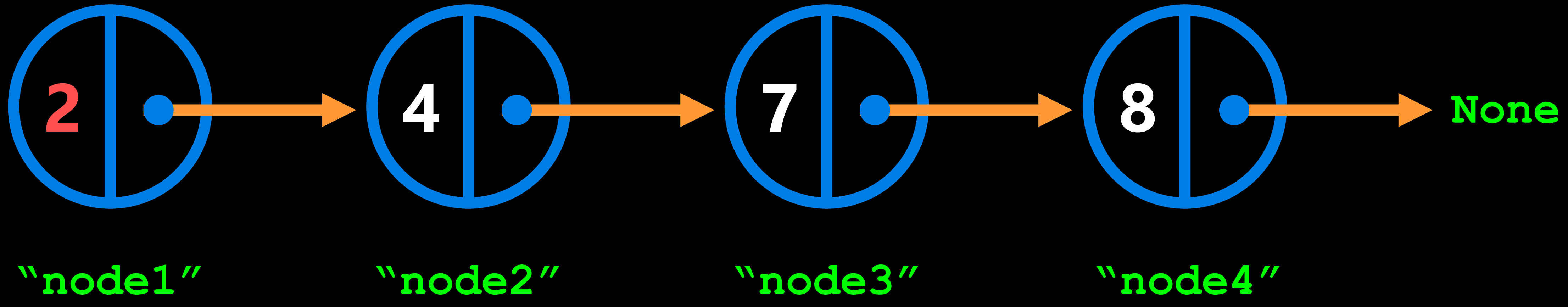


Linked Lists

- What is a linked list?

```
>>> node1.cargo
2
```

Value
Pointer



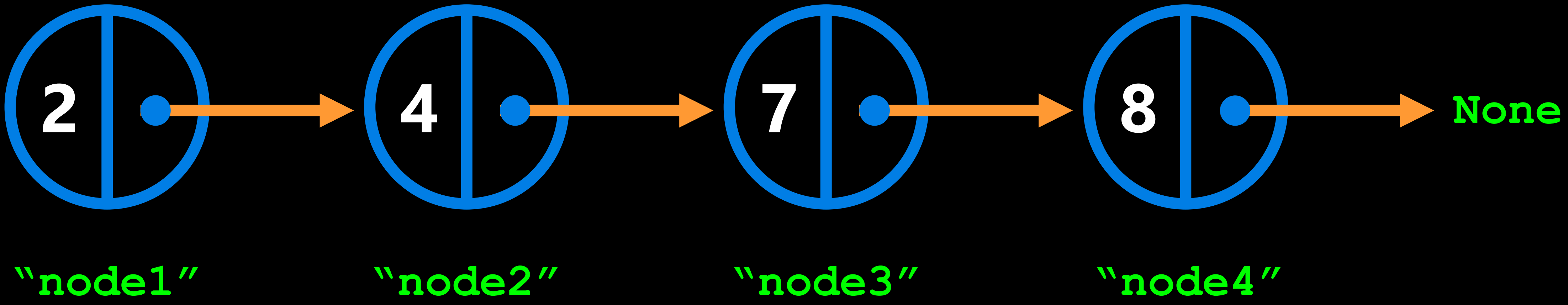
Linked Lists

- What is a linked list?

```
>>> node4.cargo
?
```

Value

Pointer

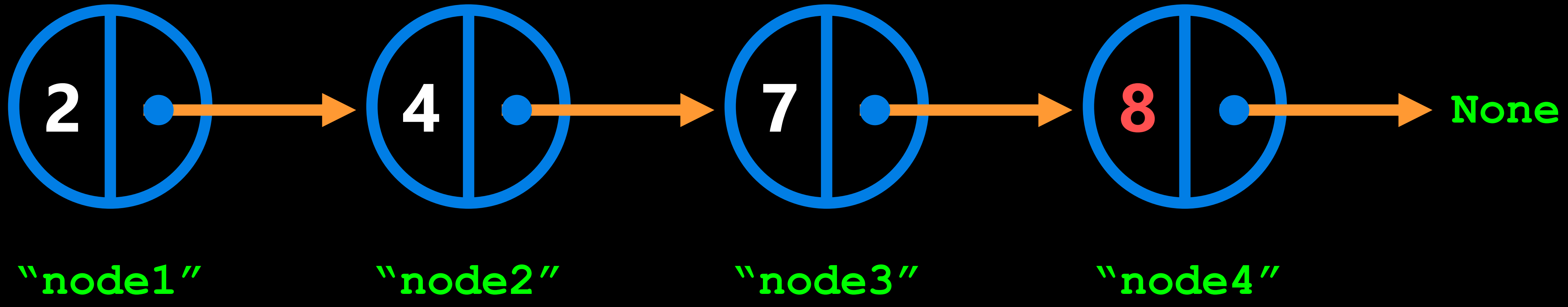


Linked Lists

- What is a linked list?

```
>>> node4.cargo
8
```

Value
Pointer



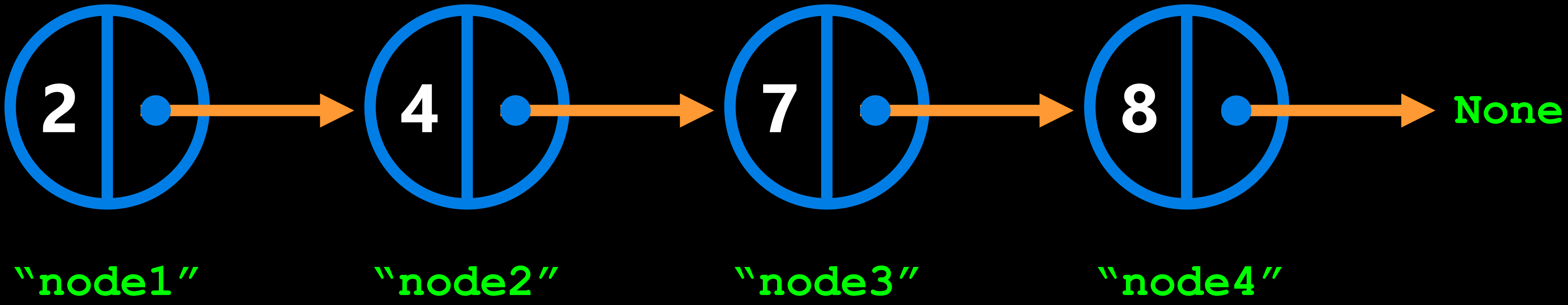
Linked Lists

- What is a linked list?

```
>>> node2.next
?
```

Value

Pointer



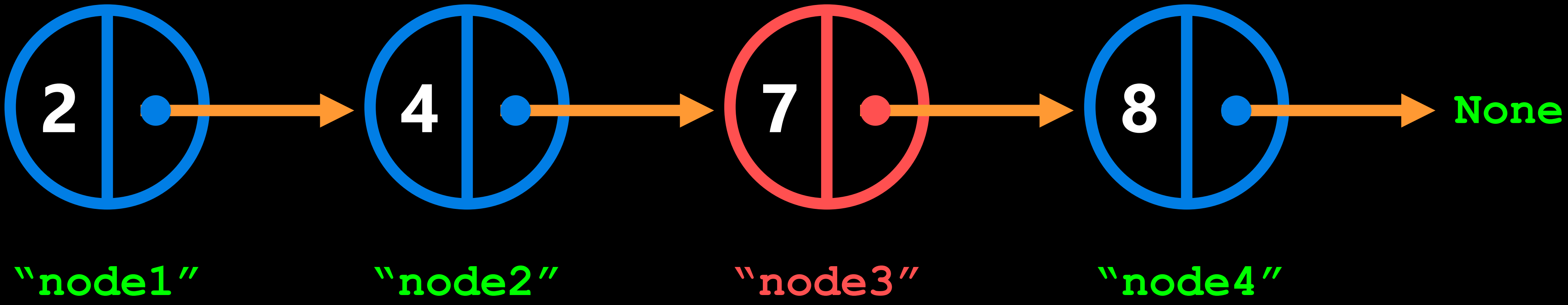
Linked Lists

- What is a linked list?

```
>>> node2.next
node3
```

Value

Pointer

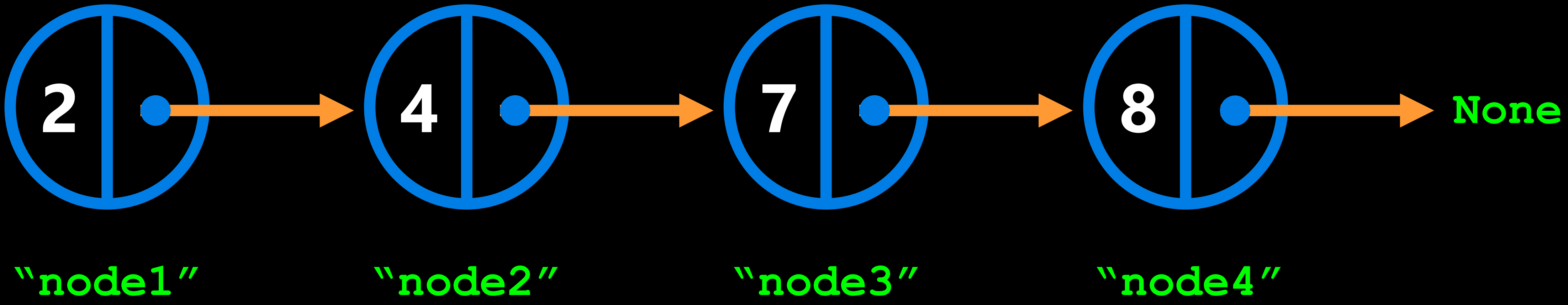


Linked Lists

- What is a linked list?

```
>>> node4.next
?
```

Value
Pointer

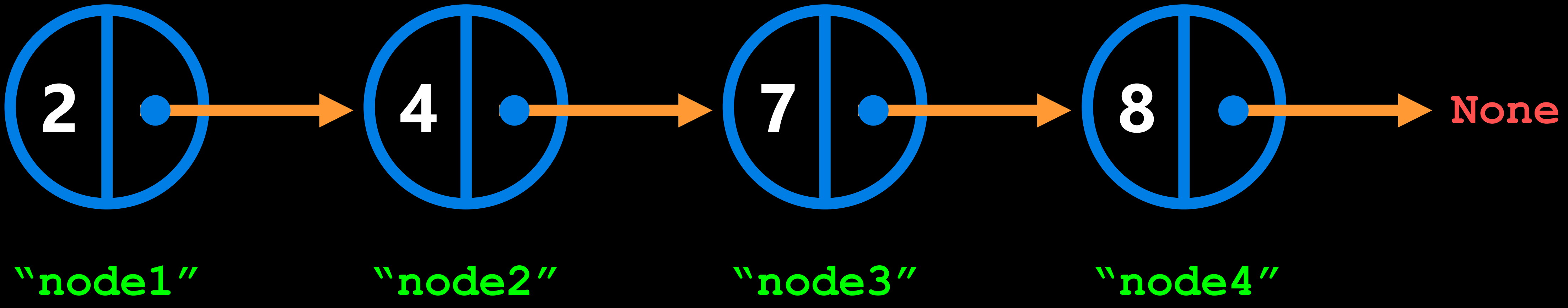


Linked Lists

- What is a linked list?

```
>>> node4.next
None
```

Value
Pointer

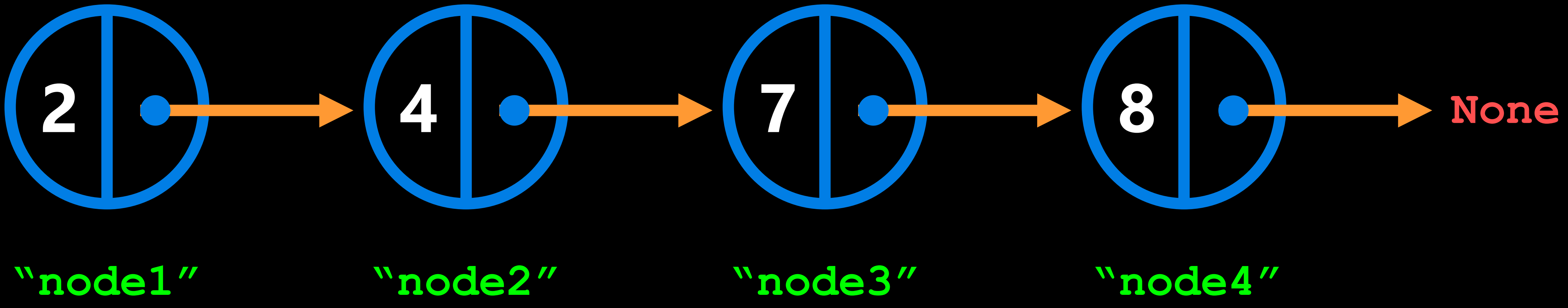


Linked Lists

- What is a linked list?

```
>>> node4.next = Node(3)

>>> node4.next
?
```

Value
Pointer

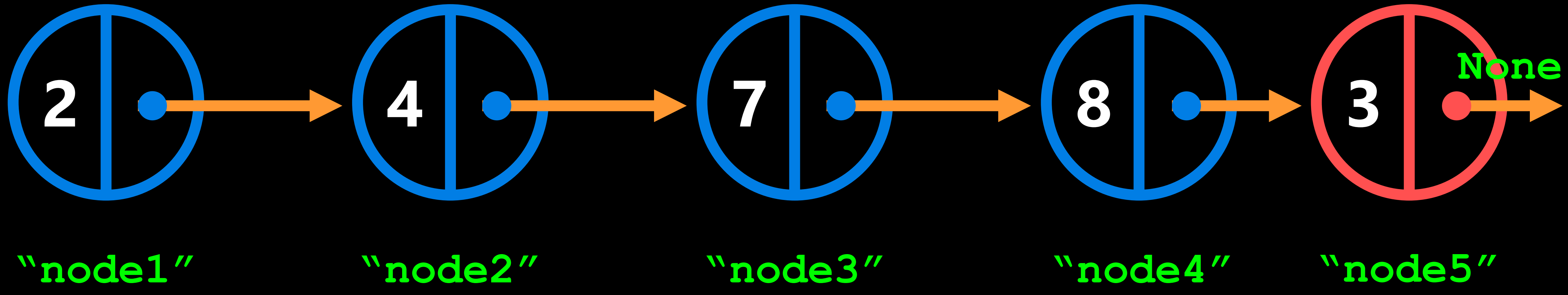


Linked Lists

- What is a linked list?

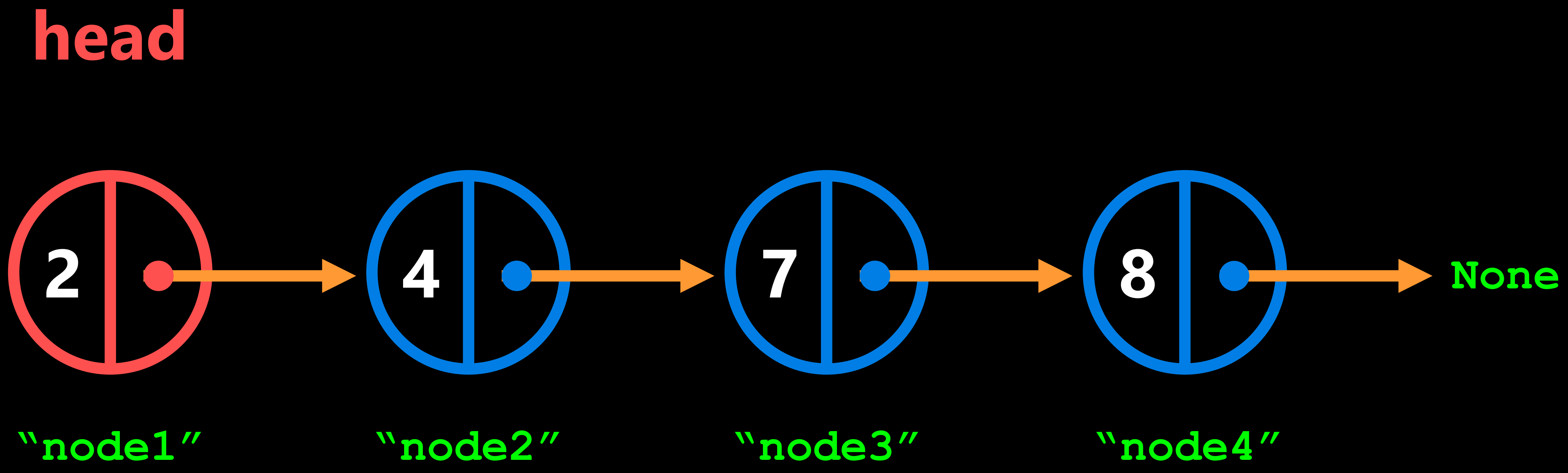
```
>>> node4.next = Node(3)

>>> node4.next
node5
```

Value
Pointer



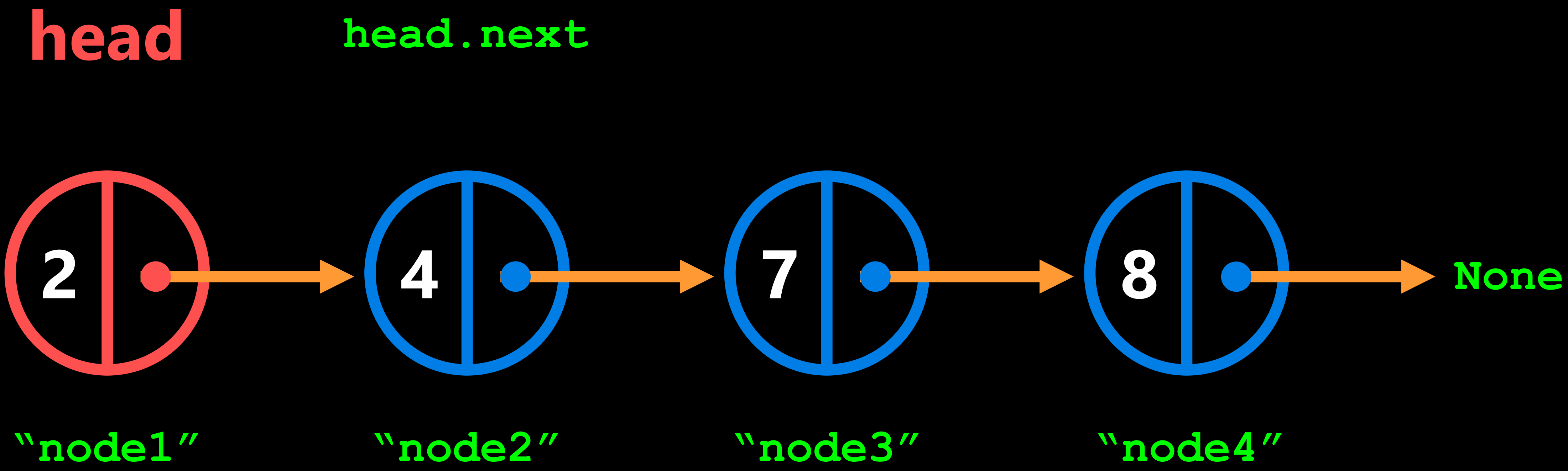
Linked Lists

- What is a linked list?



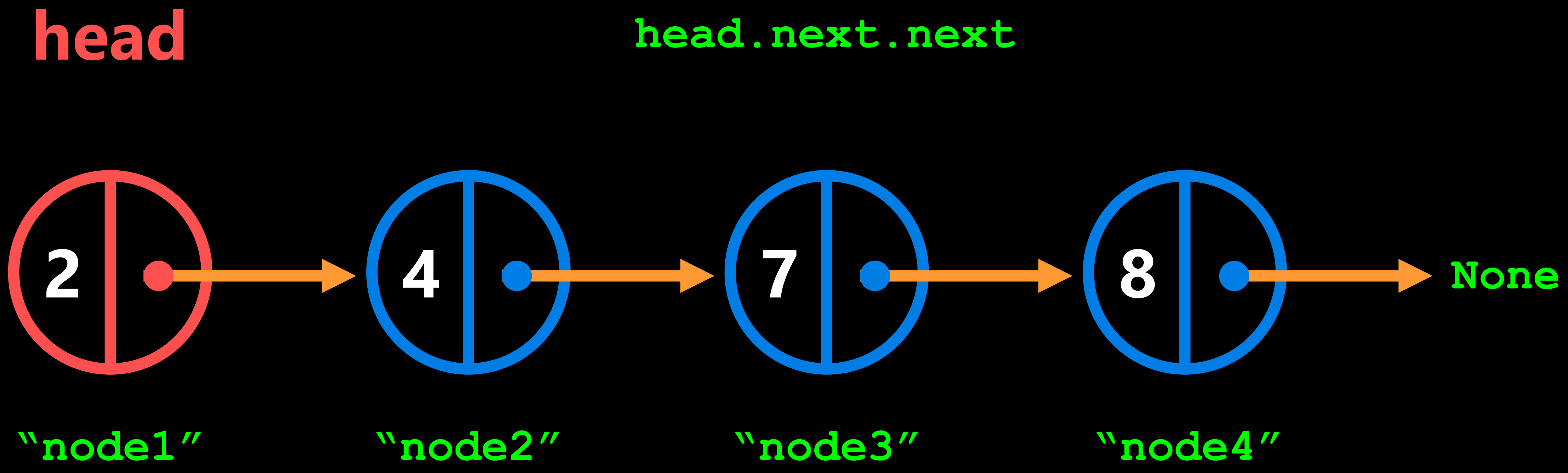
Linked Lists

- What is a linked list?



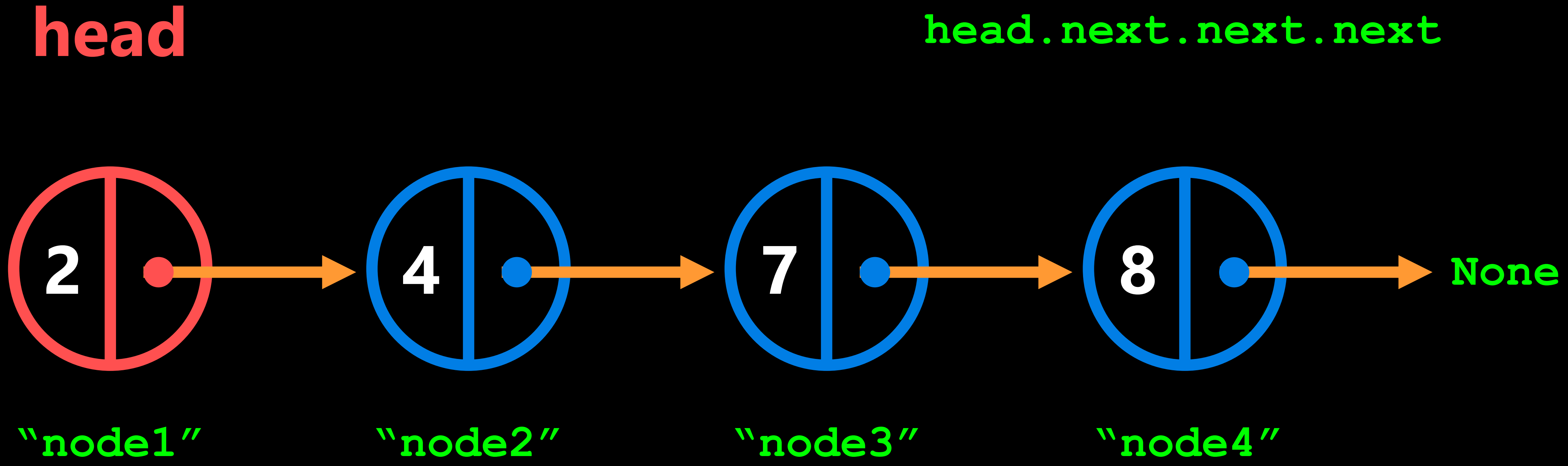
Linked Lists

- What is a linked list?



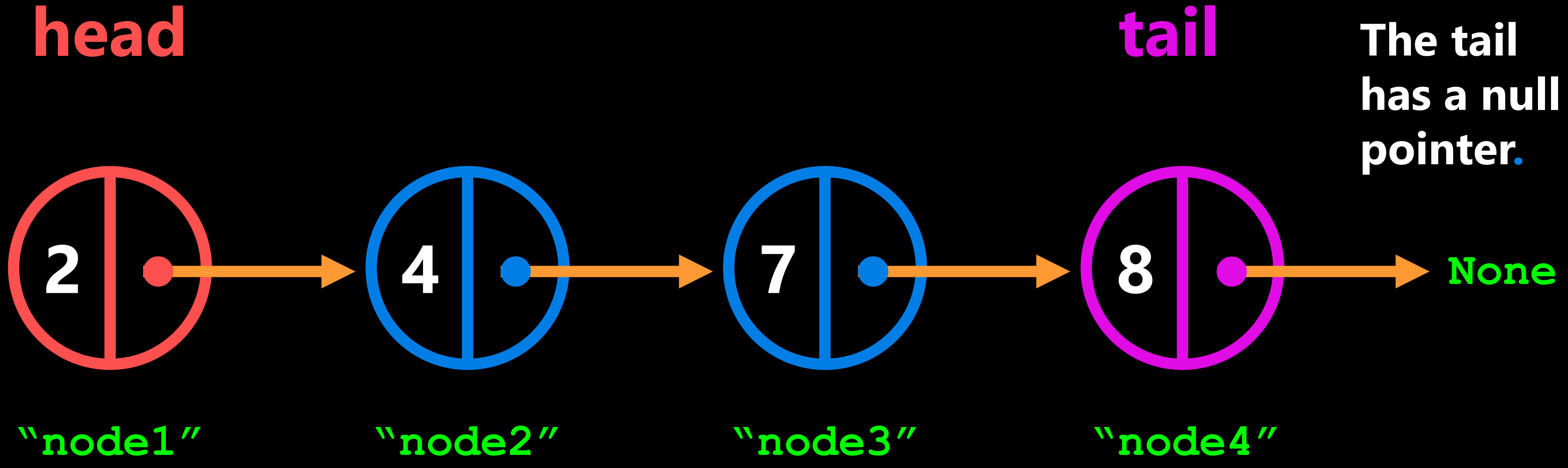
Linked Lists

- What is a linked list?



Linked Lists

- What is a linked list?



The tail has a null pointer.

The Linked List Class

- Let's check out the LinkedList class functionality.

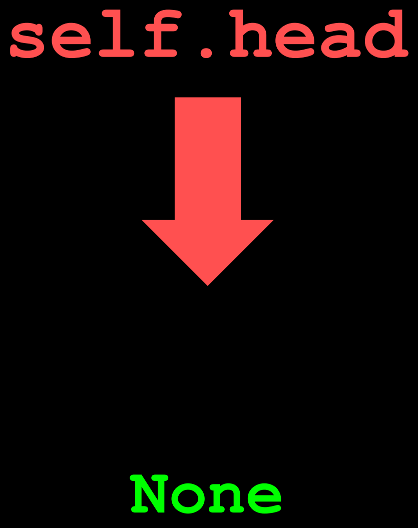
Open your notebook

Click Link:

2. LinkedList Class

```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.__str__()
'empty list'
```

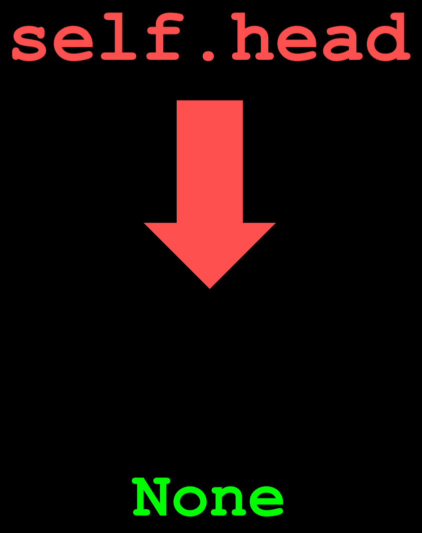


```python
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

add_to_head method.

```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.__str__()
'empty list'
```



self.head

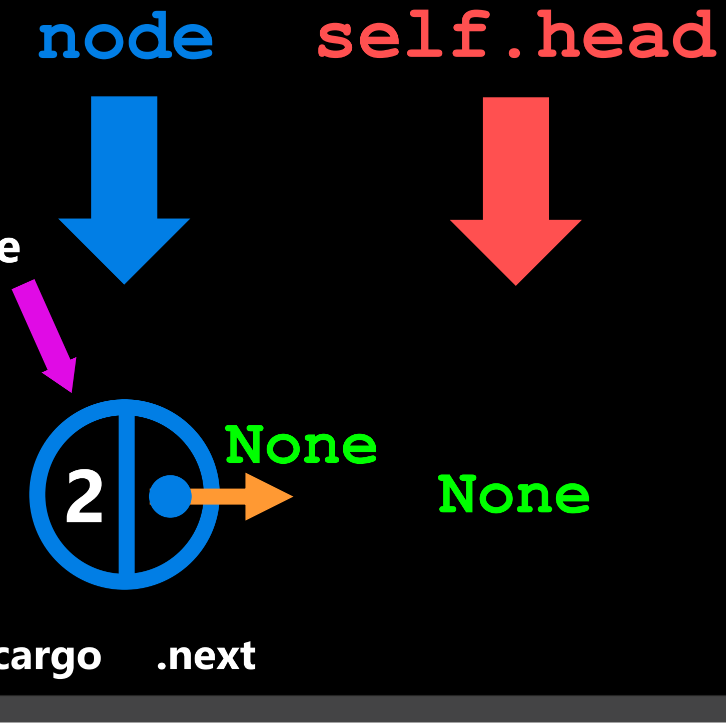
None

```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Create Node

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> None'
```

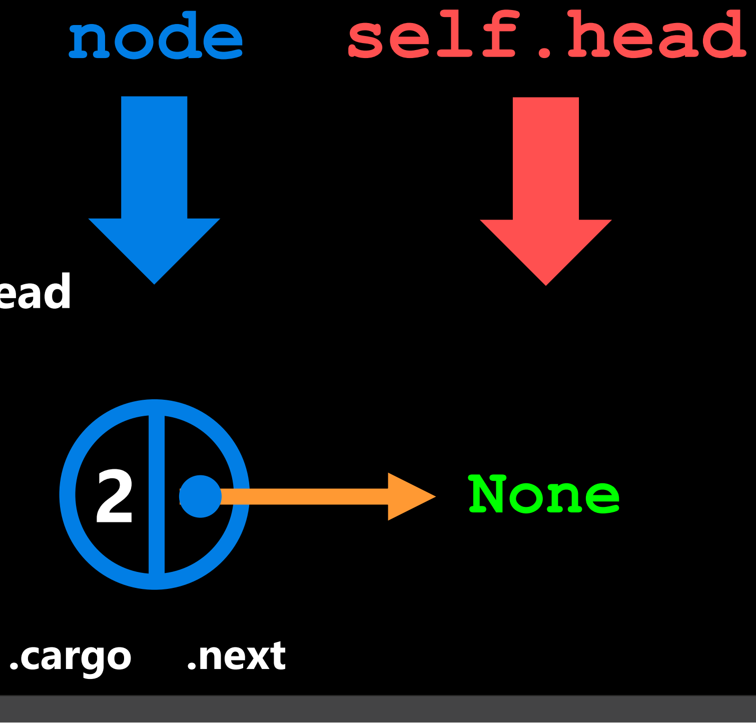
Add Node



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Point to head

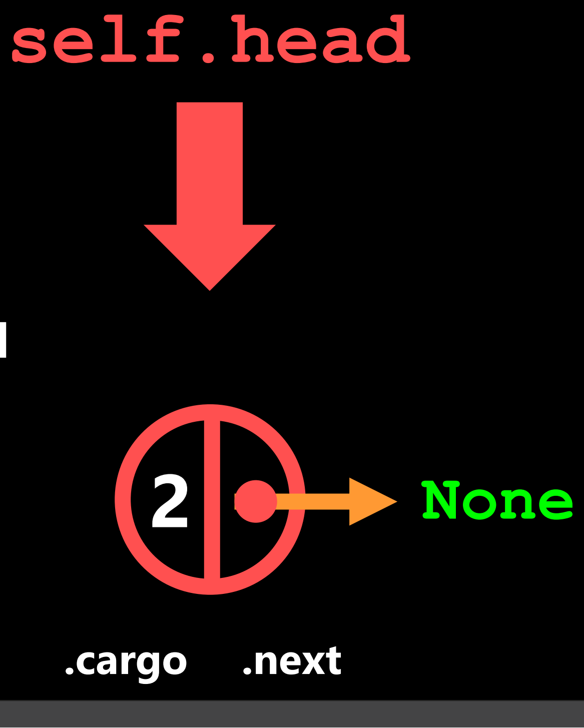
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> None'
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> None'
```

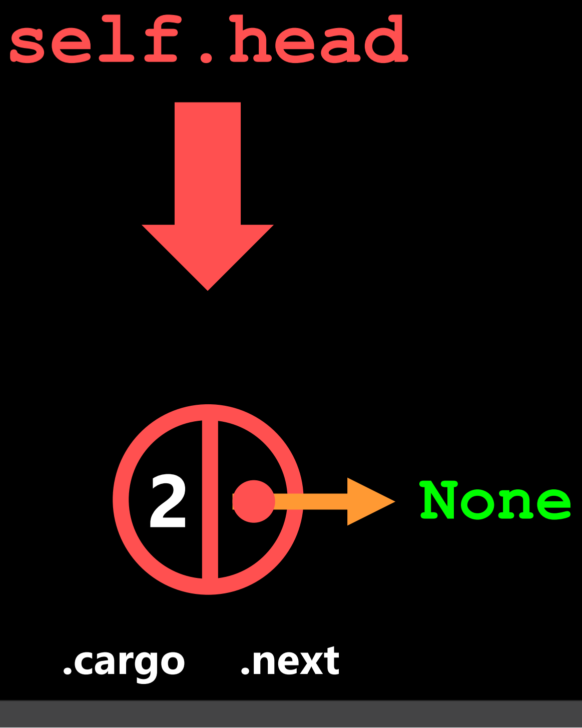
Assign new Node to head



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Increase length

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> None'
```

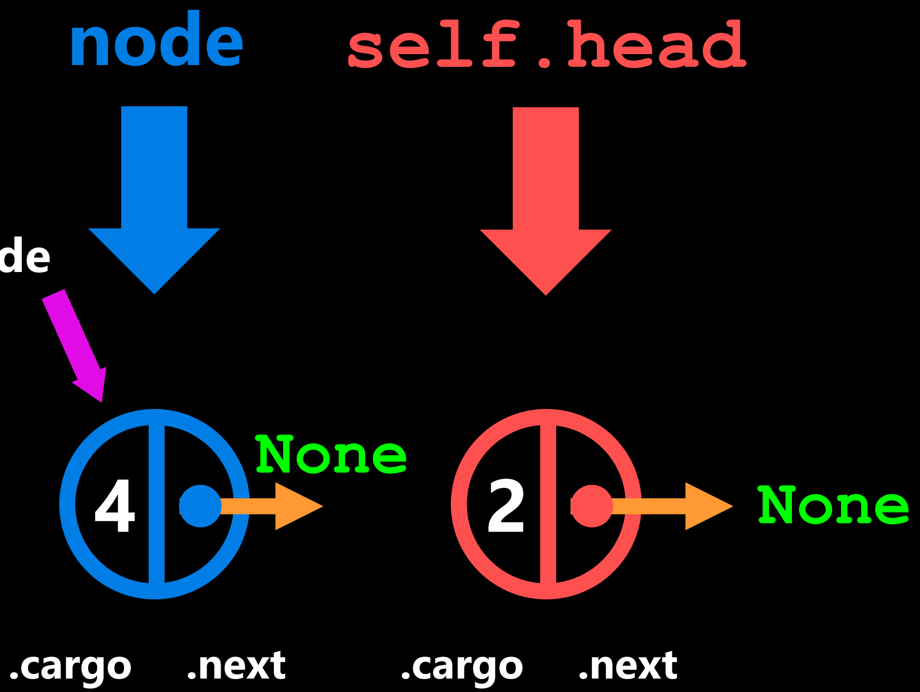


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.__str__()
'(4) --> (2) --> None'
```

Add Node

Create Node

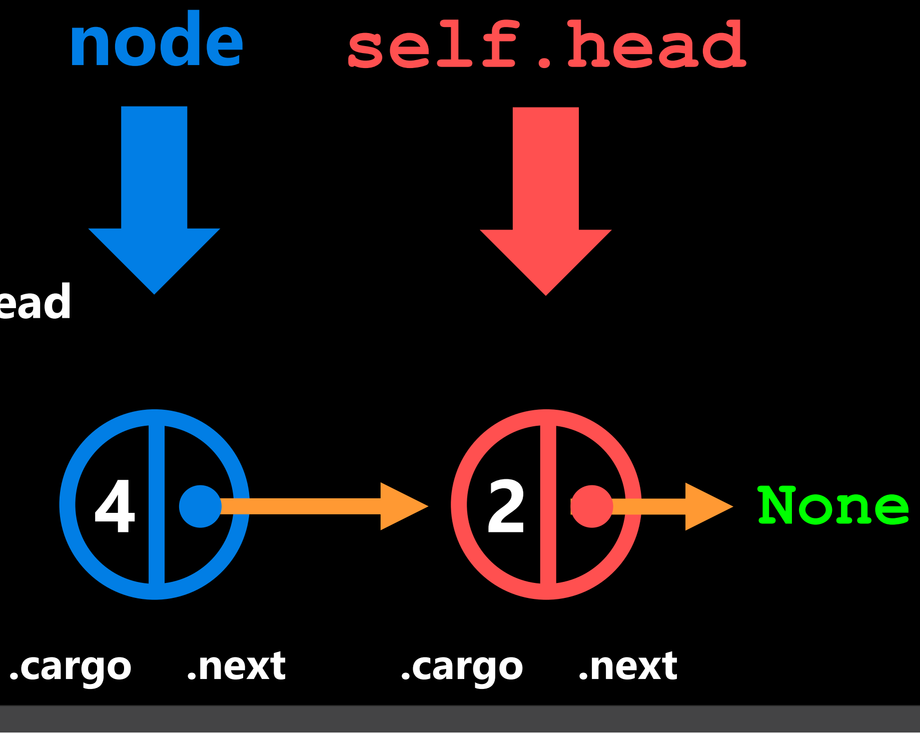


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Point to head

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.__str__()
'(4) --> (2) --> None'
```

Add Node

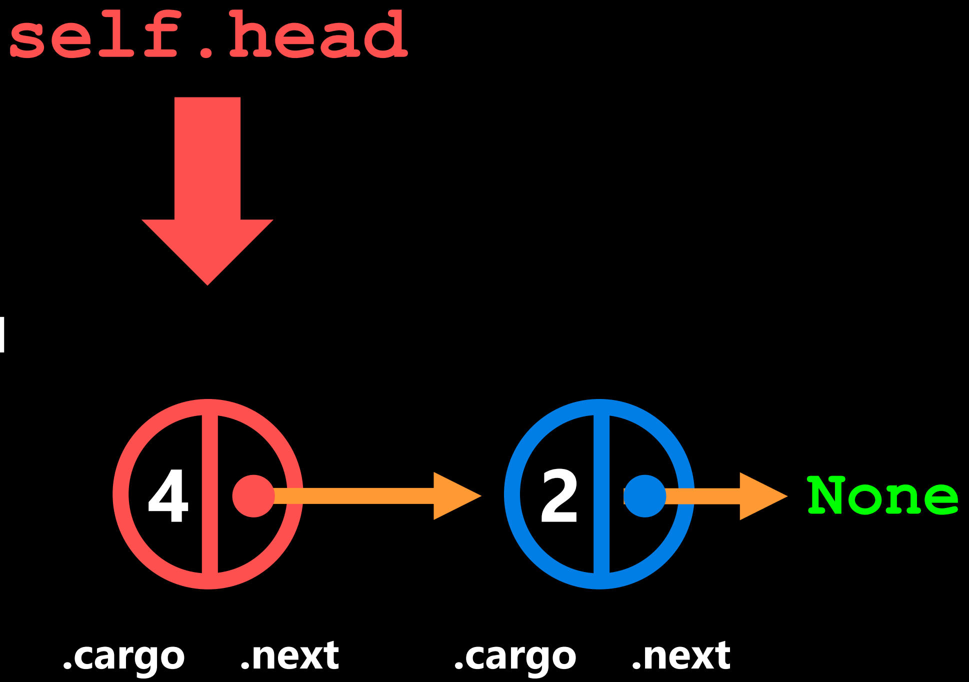


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Assign new Node to head

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.__str__()
'(4) --> (2) --> None'
```

Add Node

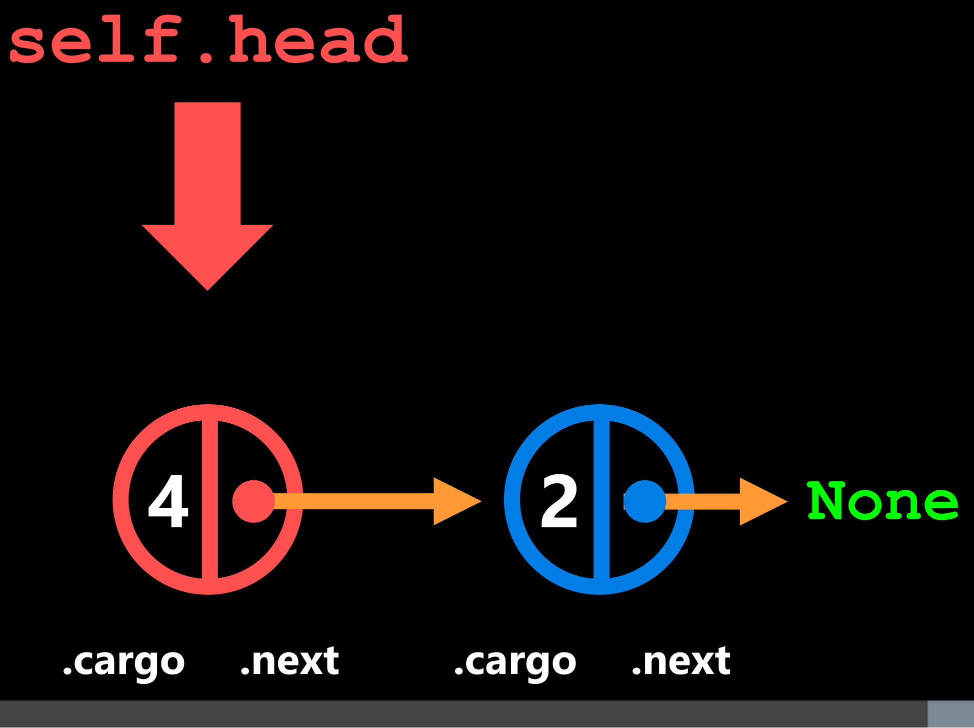


```python
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Increase length

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.__str__()
'(4) --> (2) --> None'
```

Add Node

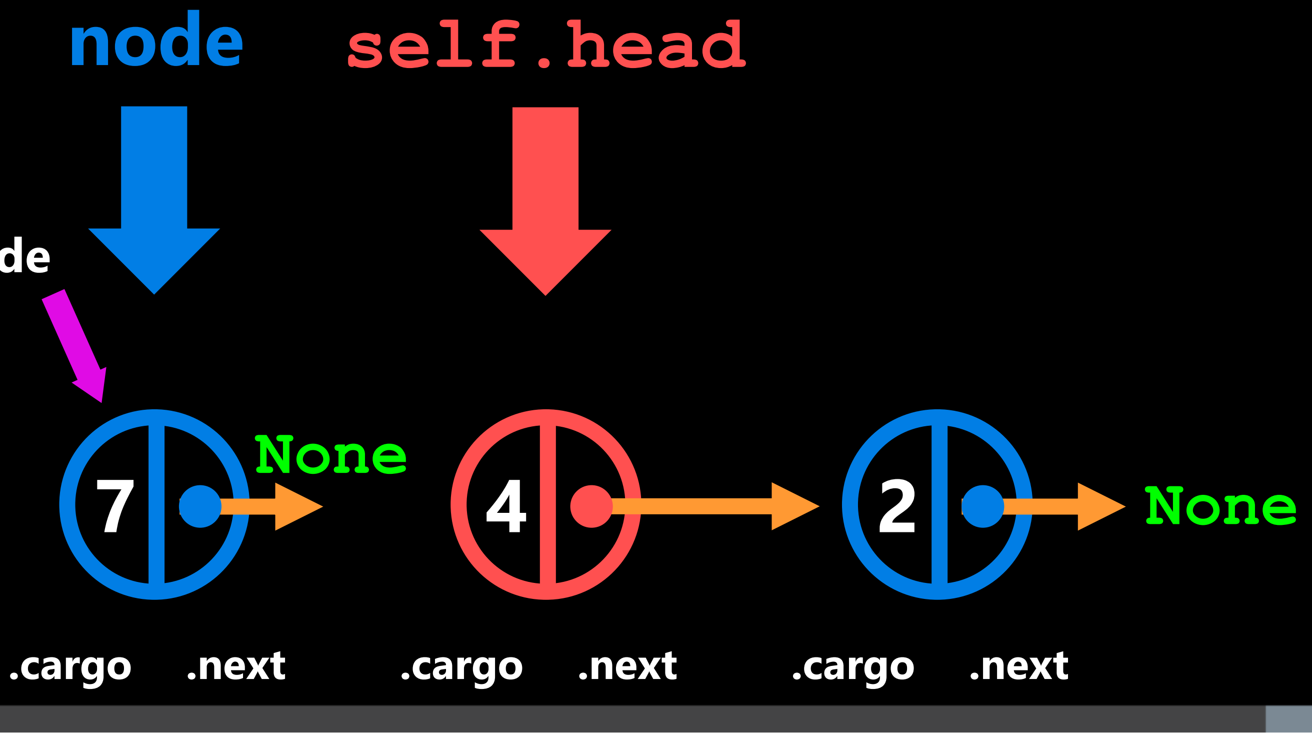


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.add_to_head(7)
>>> linked_list.__str__()
'(7) --> (4) --> (2) --> None'
```

Add Node

Create Node

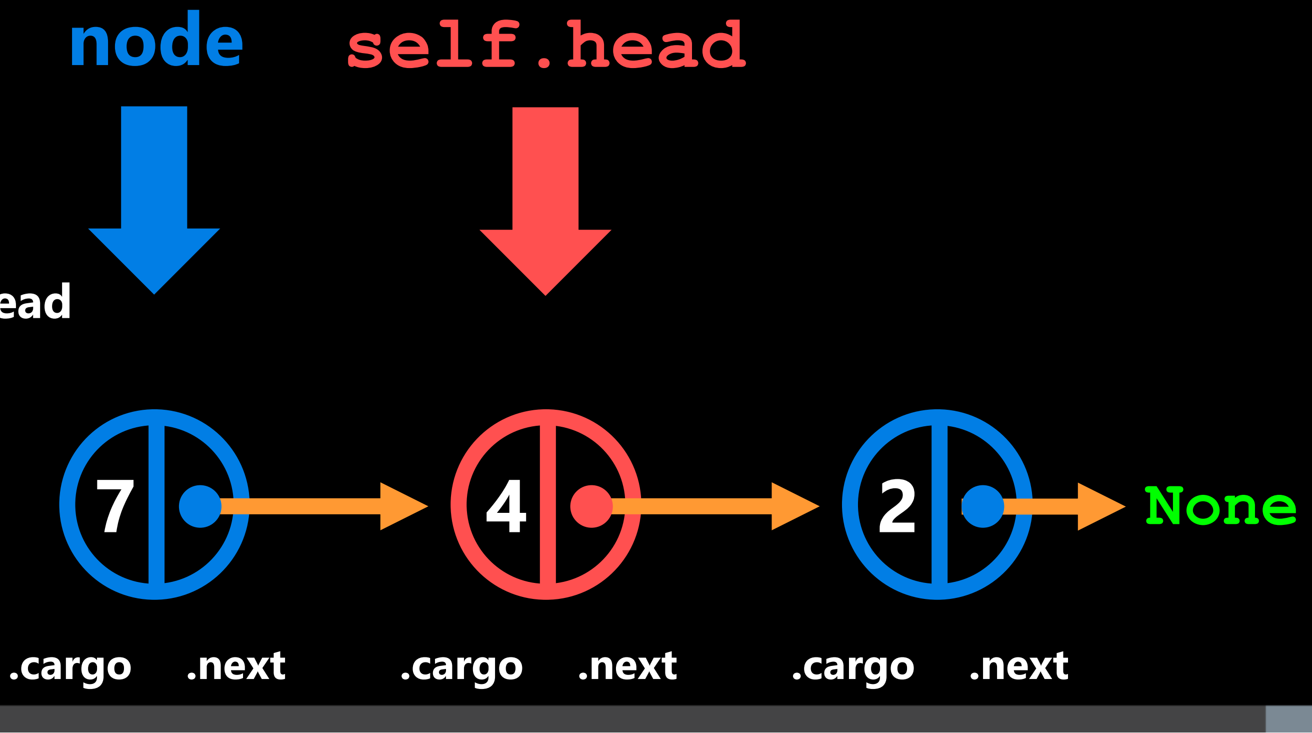


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Point to head

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.add_to_head(7)
>>> linked_list.__str__()
'(7) --> (4) --> (2) --> None'
```

Add Node

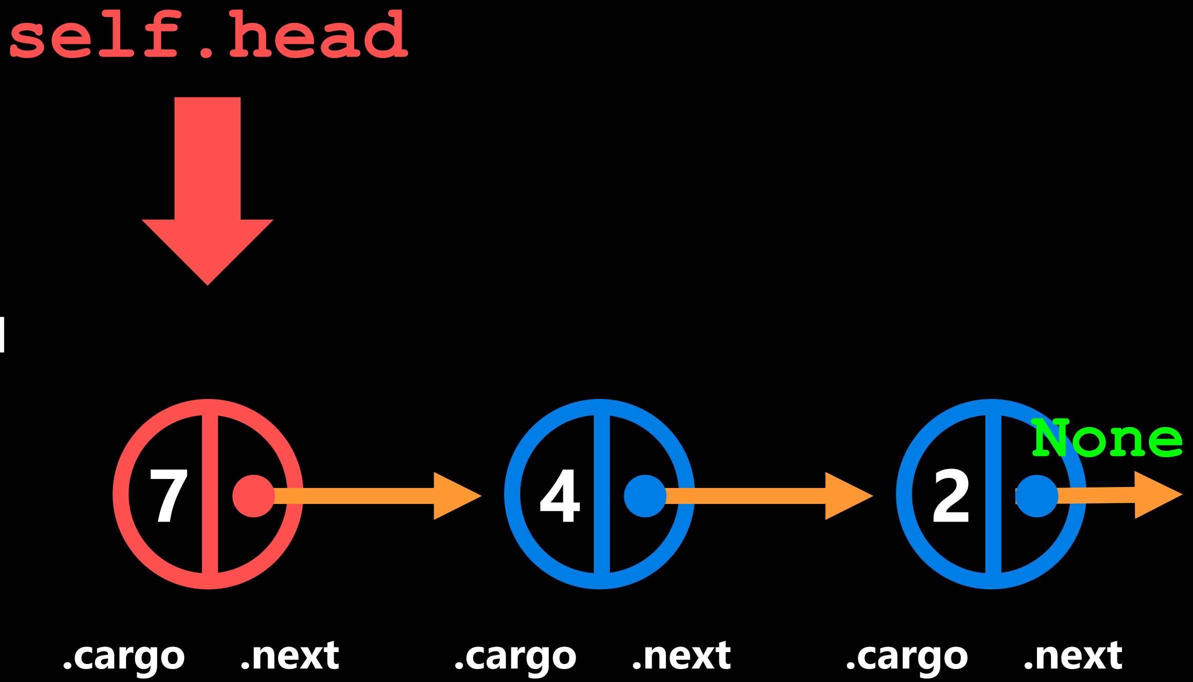


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Assign new Node to head

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.add_to_head(7)
>>> linked_list.__str__()
'(7) --> (4) --> (2) --> None'
```

Add Node

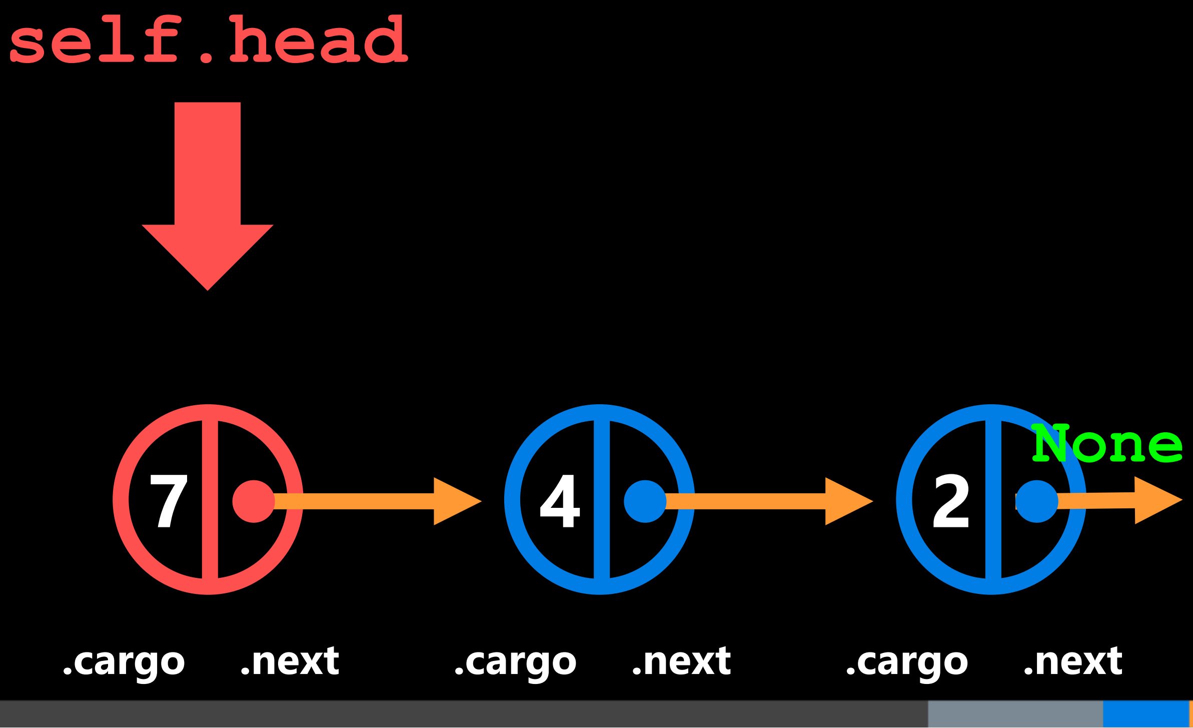


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Increase length

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(2)
>>> linked_list.add_to_head(4)
>>> linked_list.add_to_head(7)
>>> linked_list.__str__()
'(7) --> (4) --> (2) --> None'
```

Add Node

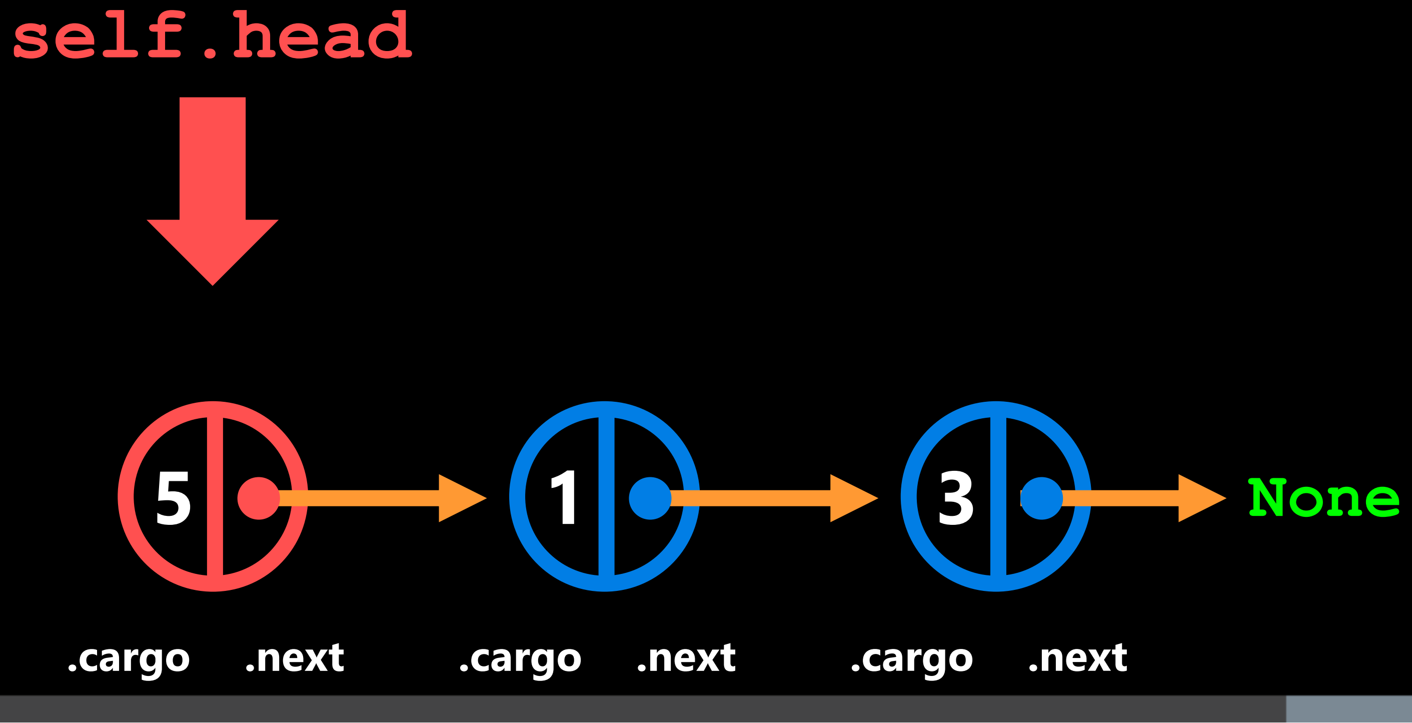


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

add_to_tail method.

```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

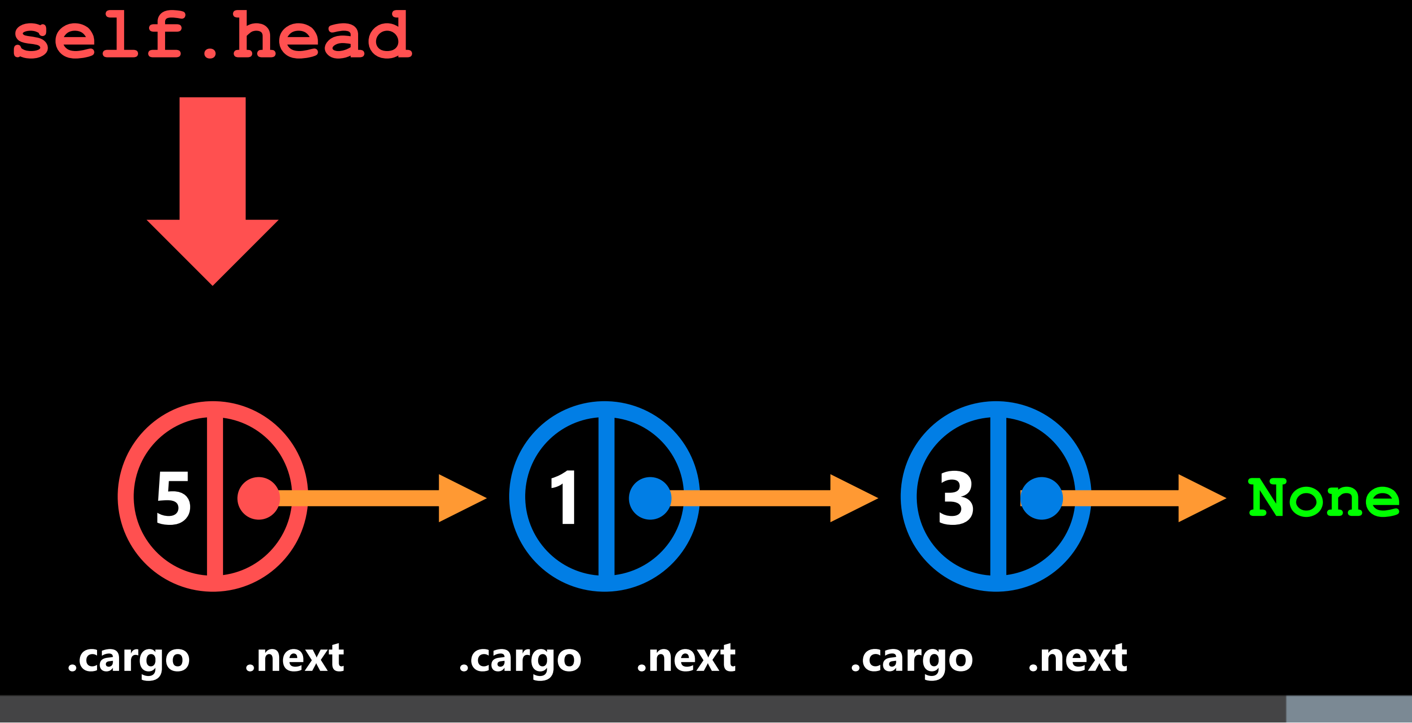
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> None'
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

Add to tail. →

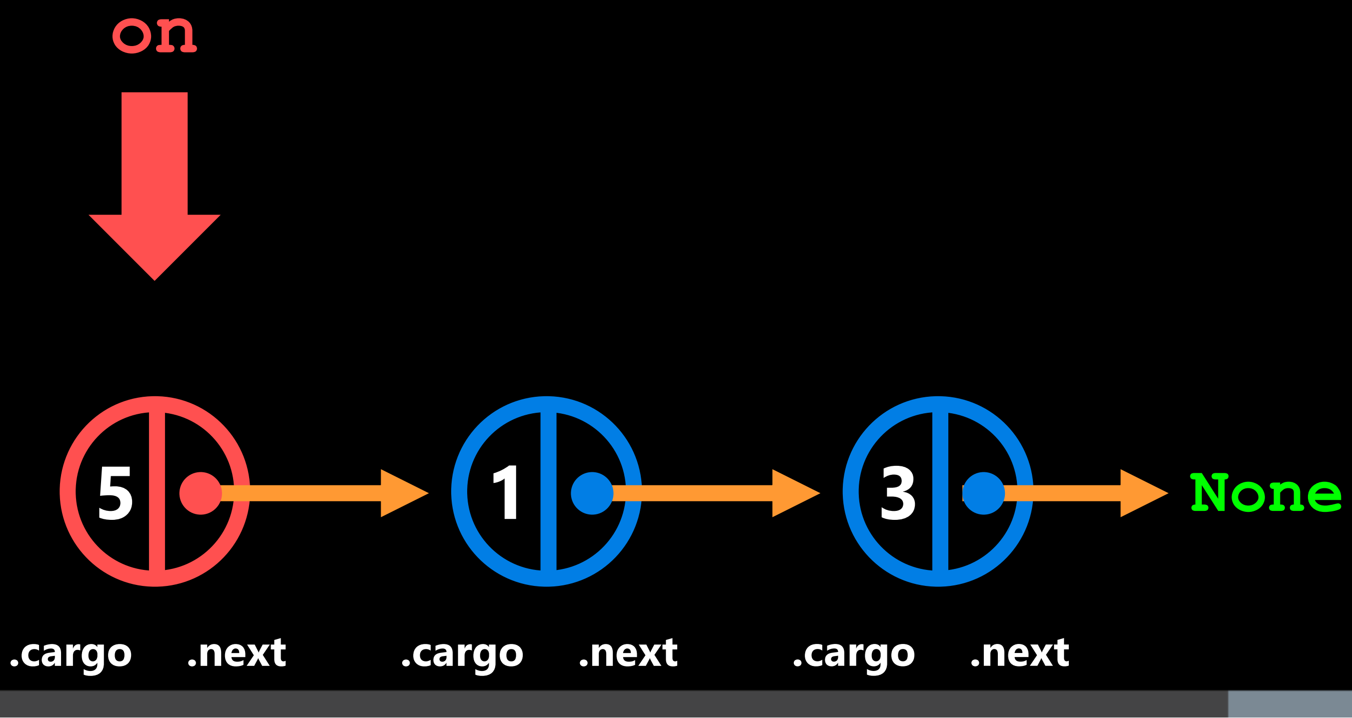


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Set on position

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

Add to tail.



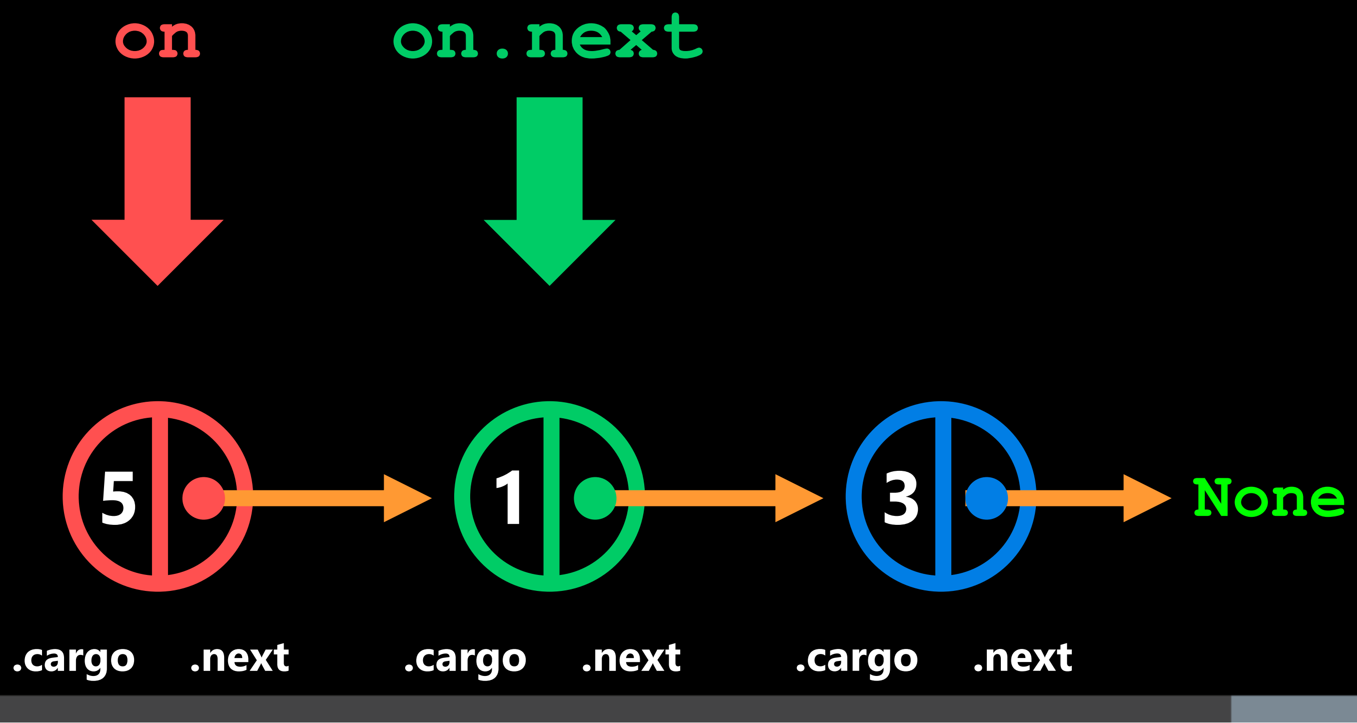
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

True

Add to tail.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

`on.next` is `None` when `on` is at the last Node.



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

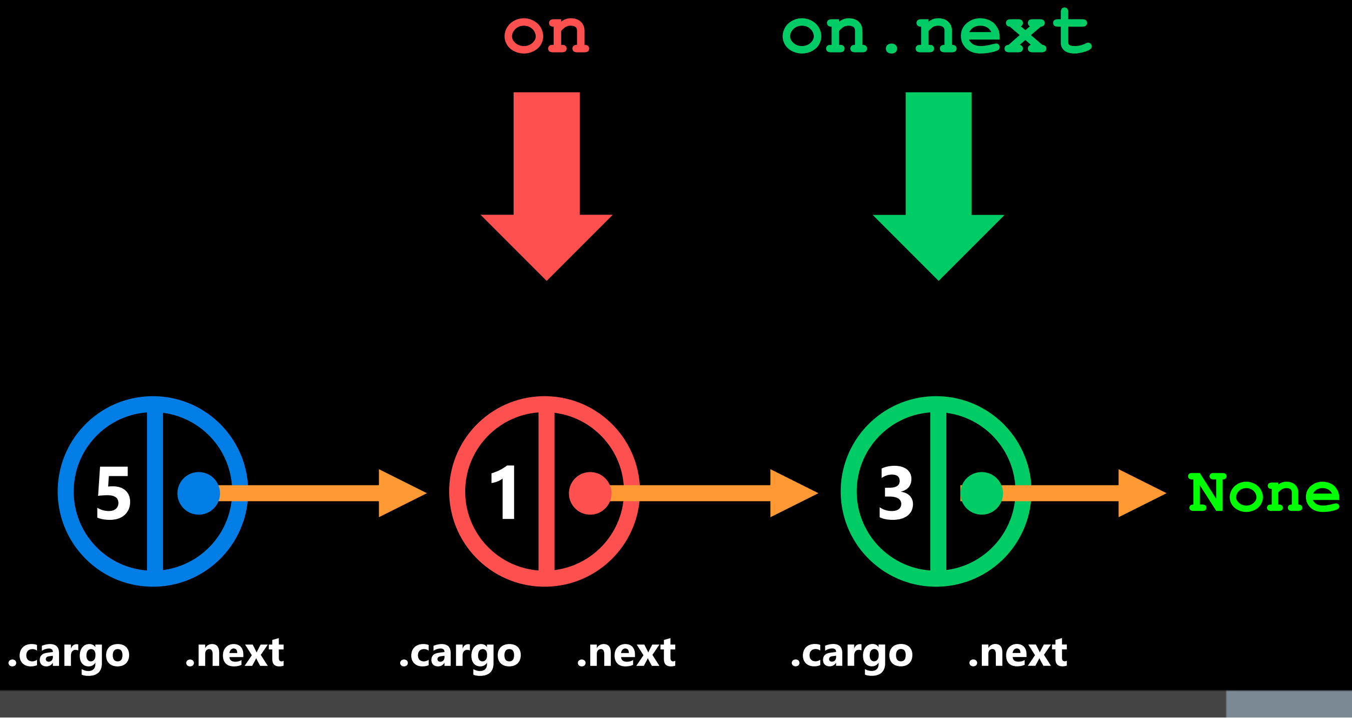
True

Move on to next position.

Add to tail.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

on.next is None when on is at the last Node.



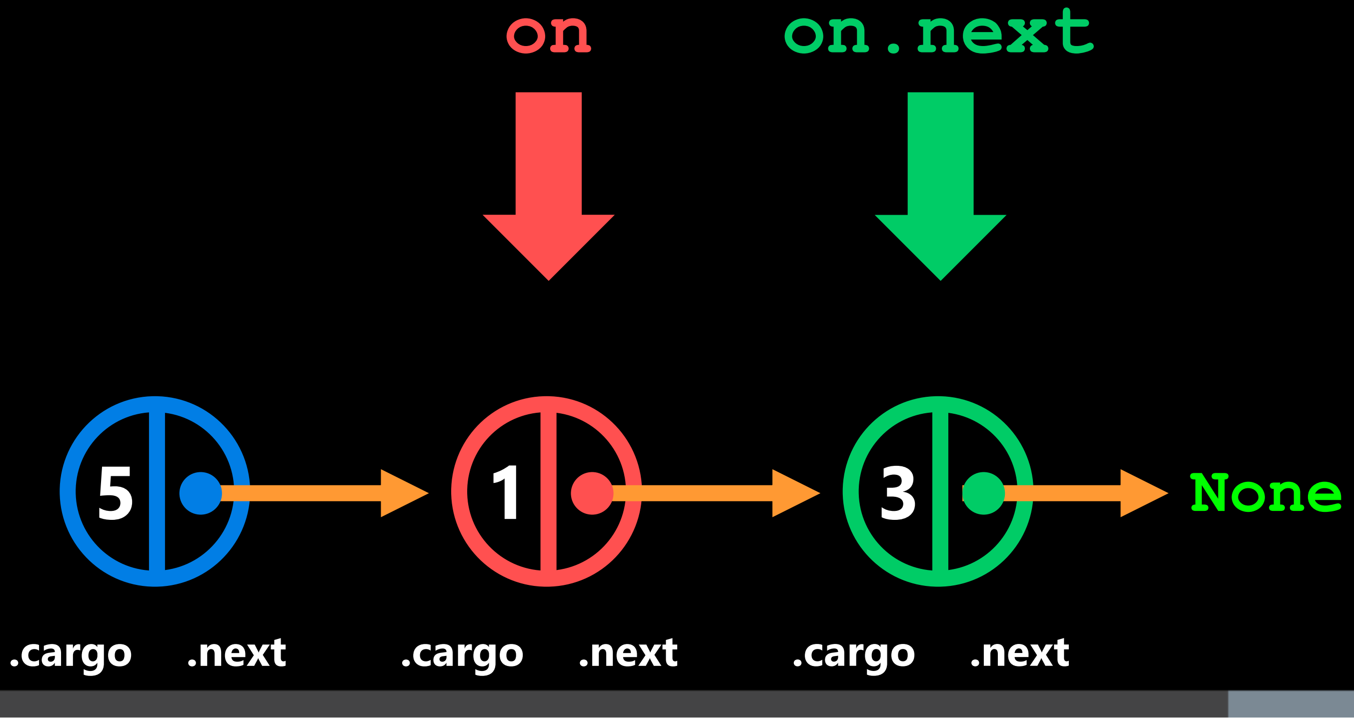
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

True

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

Add to tail.

on.next is None when on is at the last Node.



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

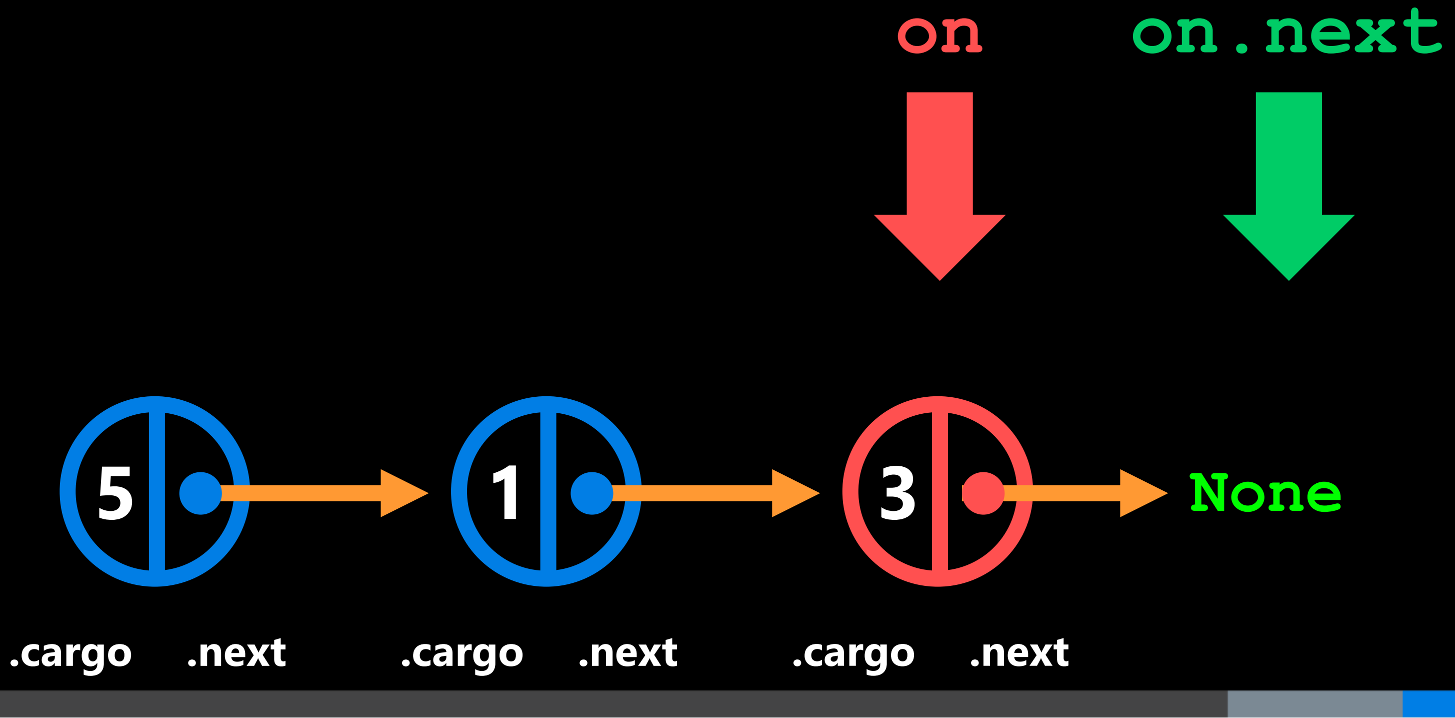
True

Move on to next position.

Add to tail.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

on.next is None when on is at the last Node.



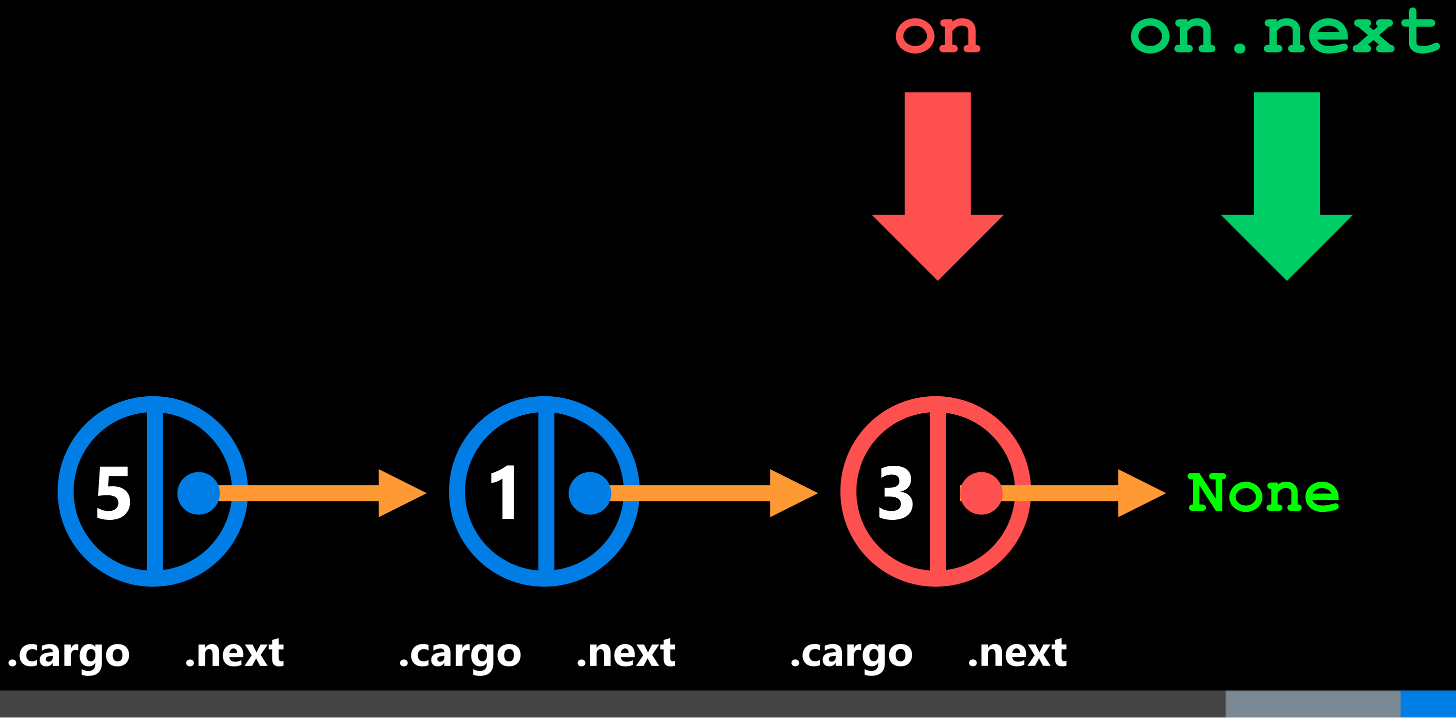
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

False

Add to tail.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

`on.next` is `None` when `on` is at the last Node.



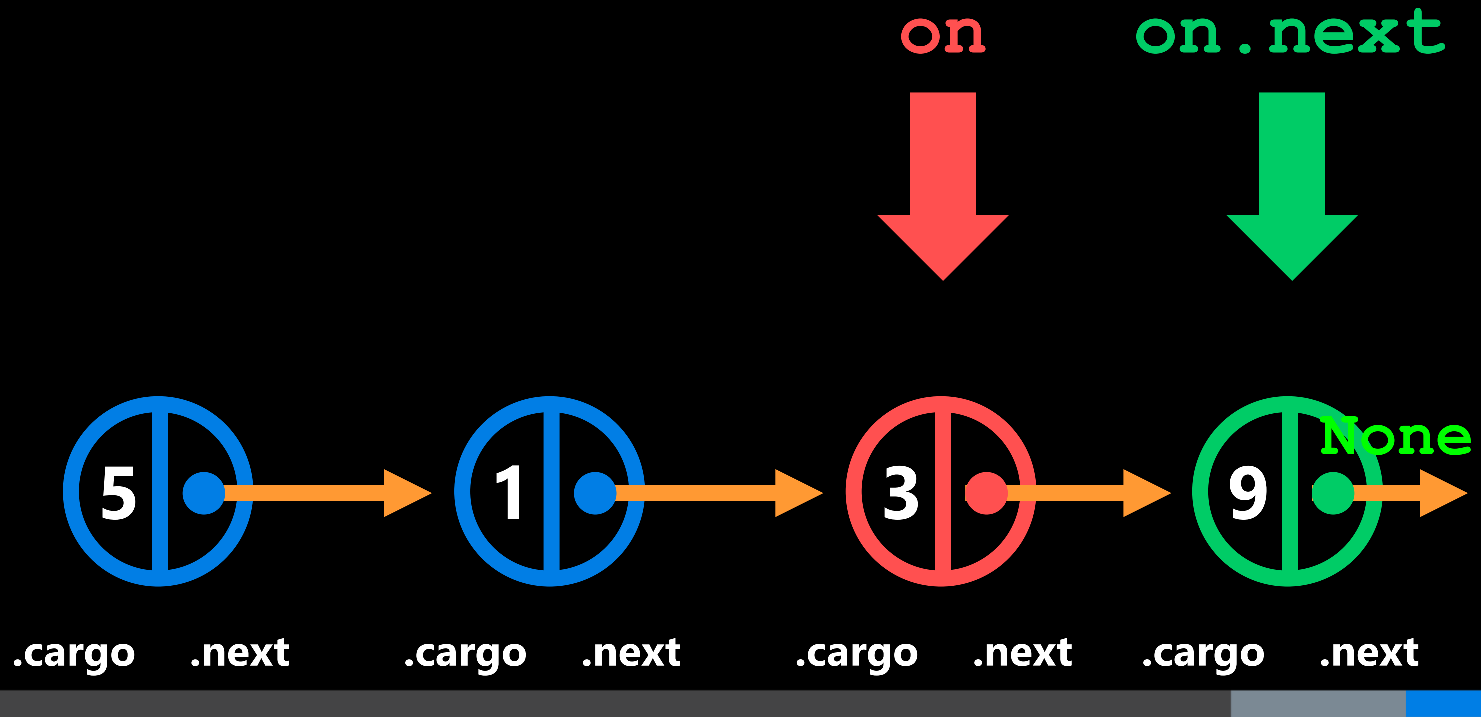
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

Add new node to tail

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(1)
>>> linked_list.add_to_head(5)
>>> linked_list.add_to_tail(9)
>>> linked_list.__str__()
'(5) --> (1) --> (3) --> (9) --> None'
```

Add to tail.

`on.next` is `None` when `on` is at the last Node.

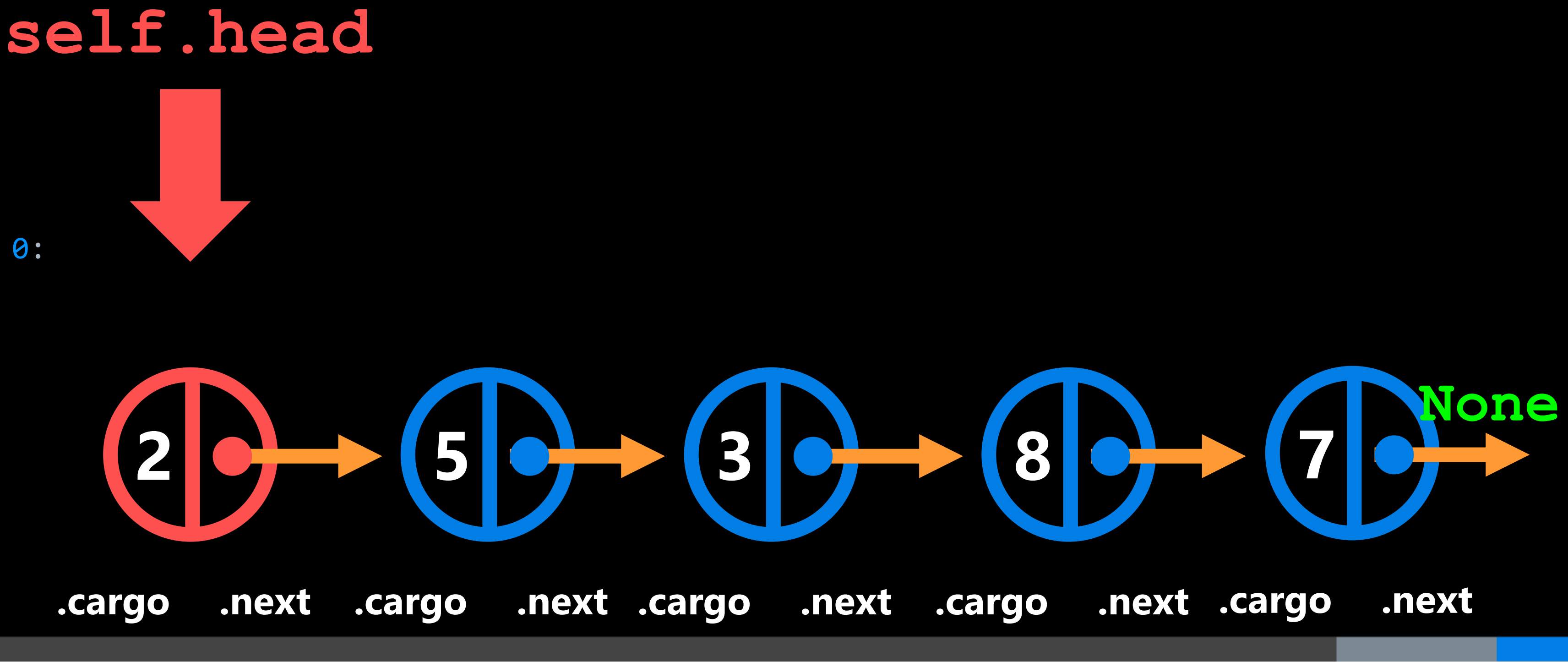


```python
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

get_at_index method.

```python
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```



```python
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

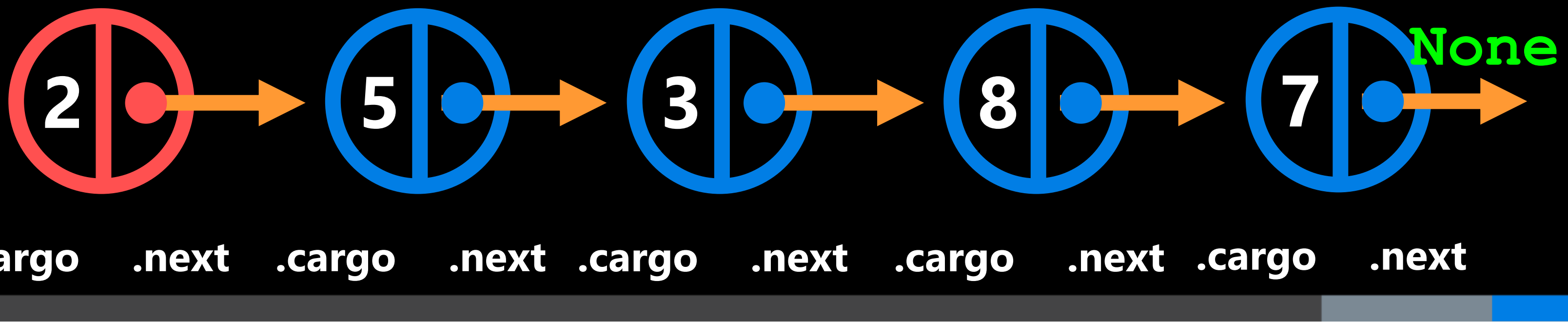
Set on position

Get node at index = 3.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.get_at_index(3)
8
```

index = 3

on



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

True ➡ while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

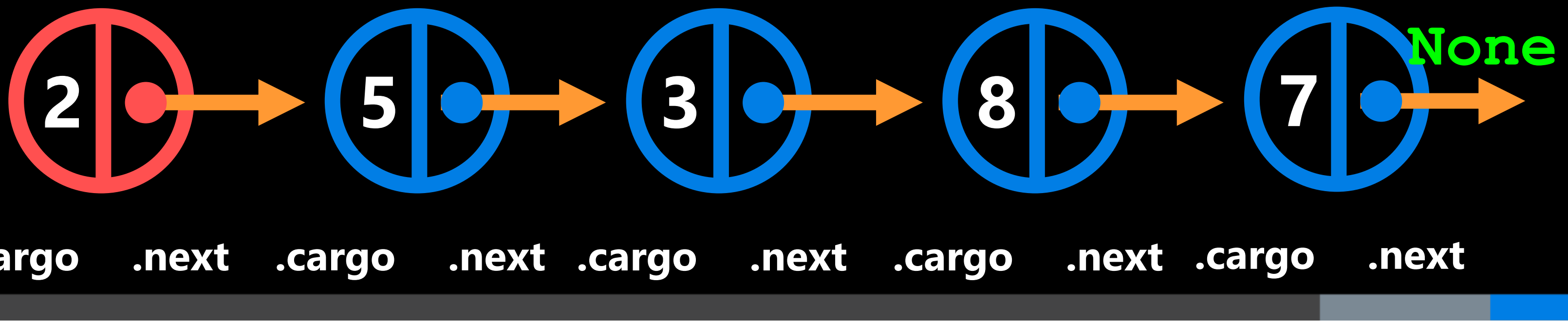
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. ➡

```
>>> linked_list.get_at_index(3)
8
```

index = 3

on



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

True → `while on is not None and index != 0:`

Move on to next position. → `on = on.next`

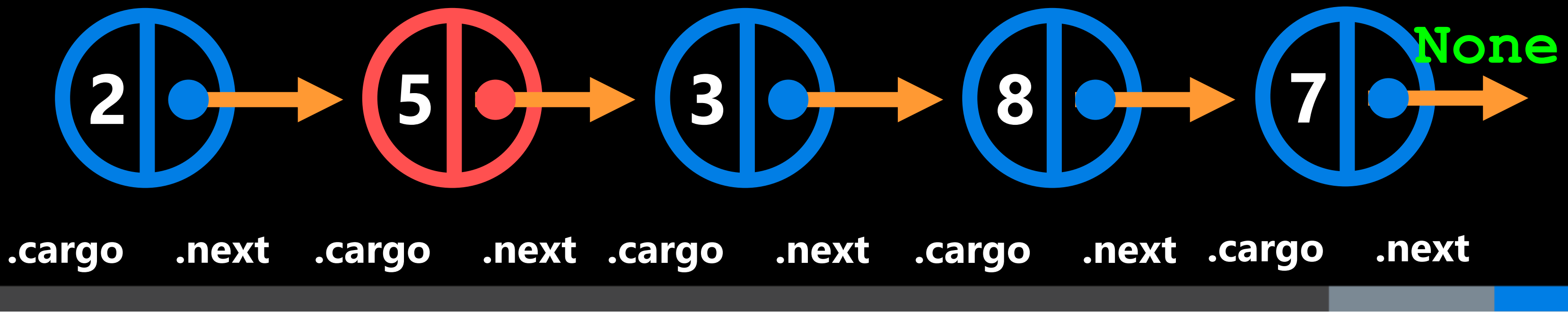
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. →

```
>>> linked_list.get_at_index(3)
8
```

index = 3

on



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

True → while

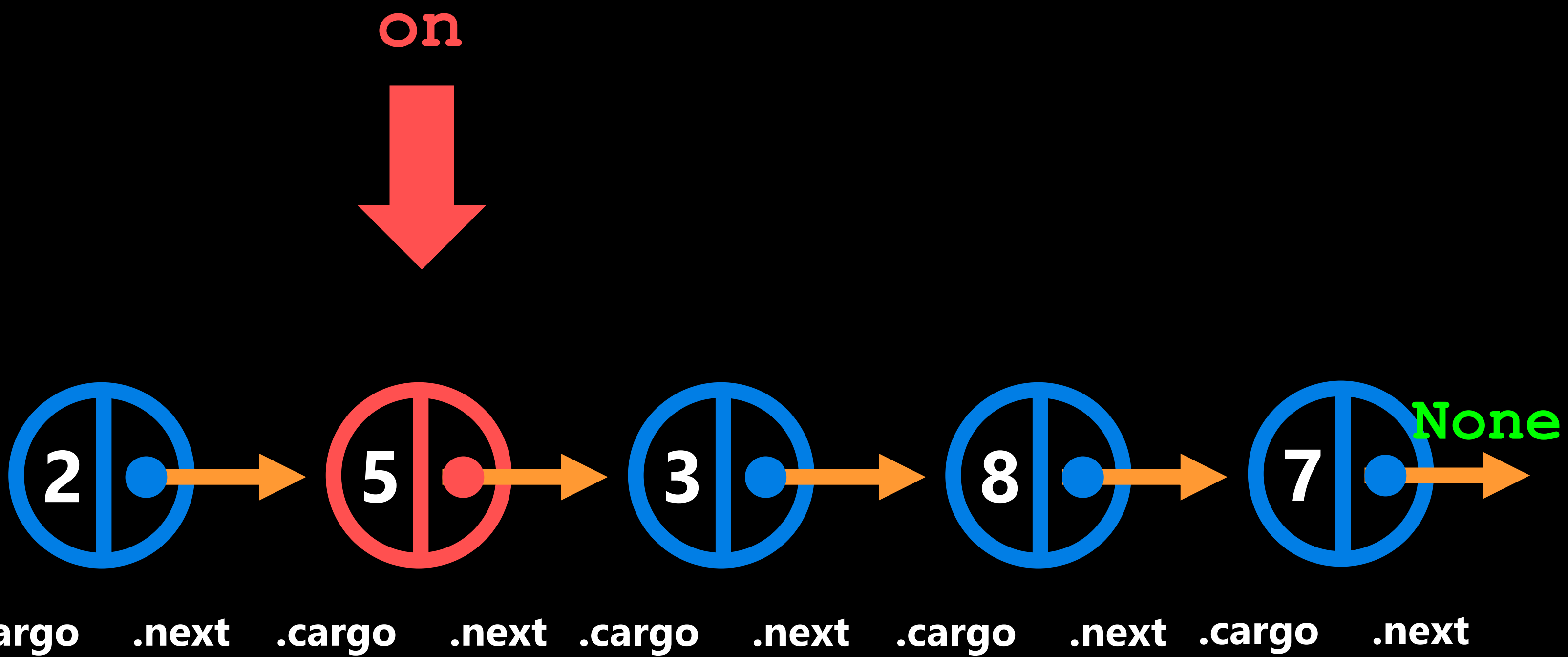
Update index.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. →

```
>>> linked_list.get_at_index(3)
8
```

index = 2

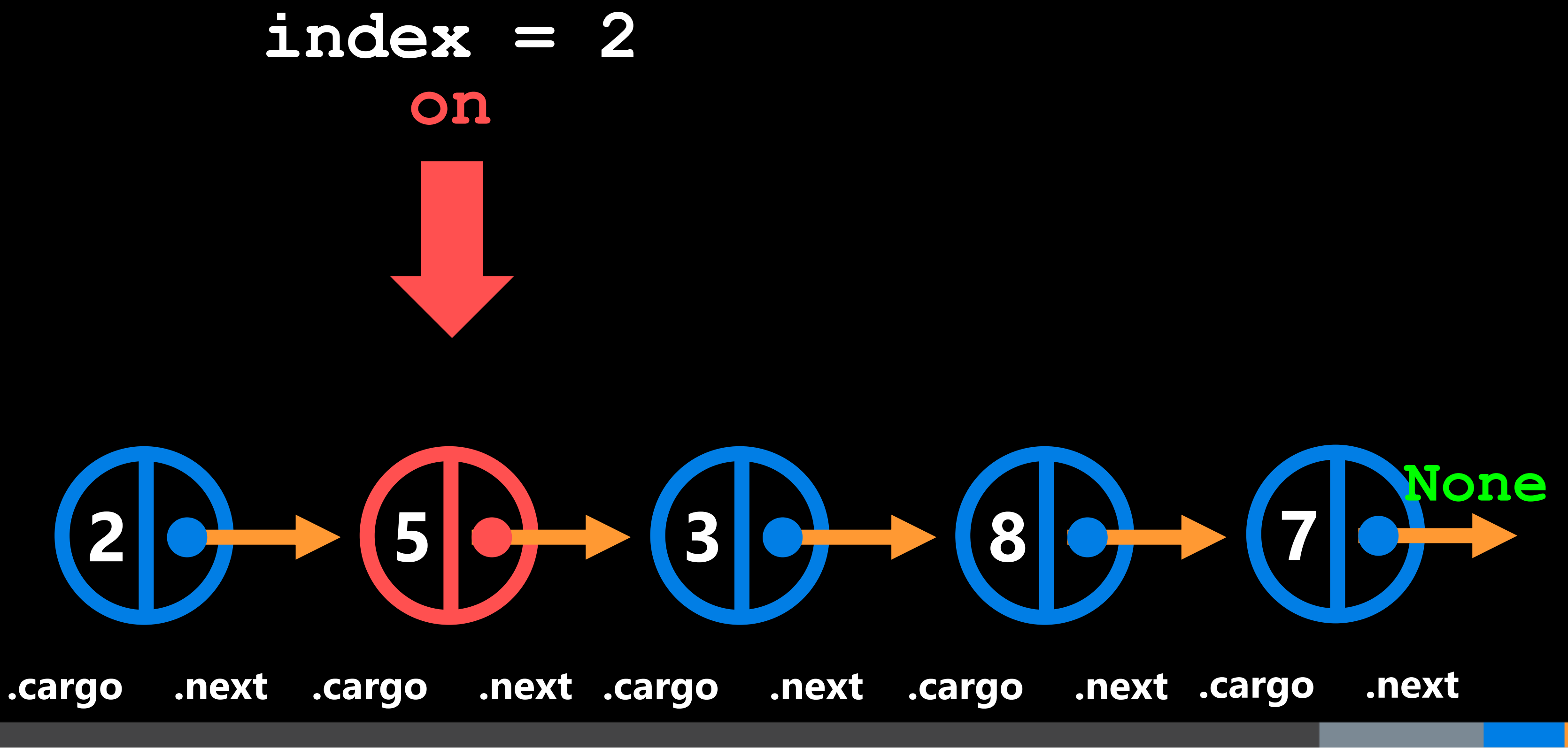


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

True ➡ (while on is not None and index != 0:)

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.get_at_index(3)
8
```

Get node at index = 3. ➡



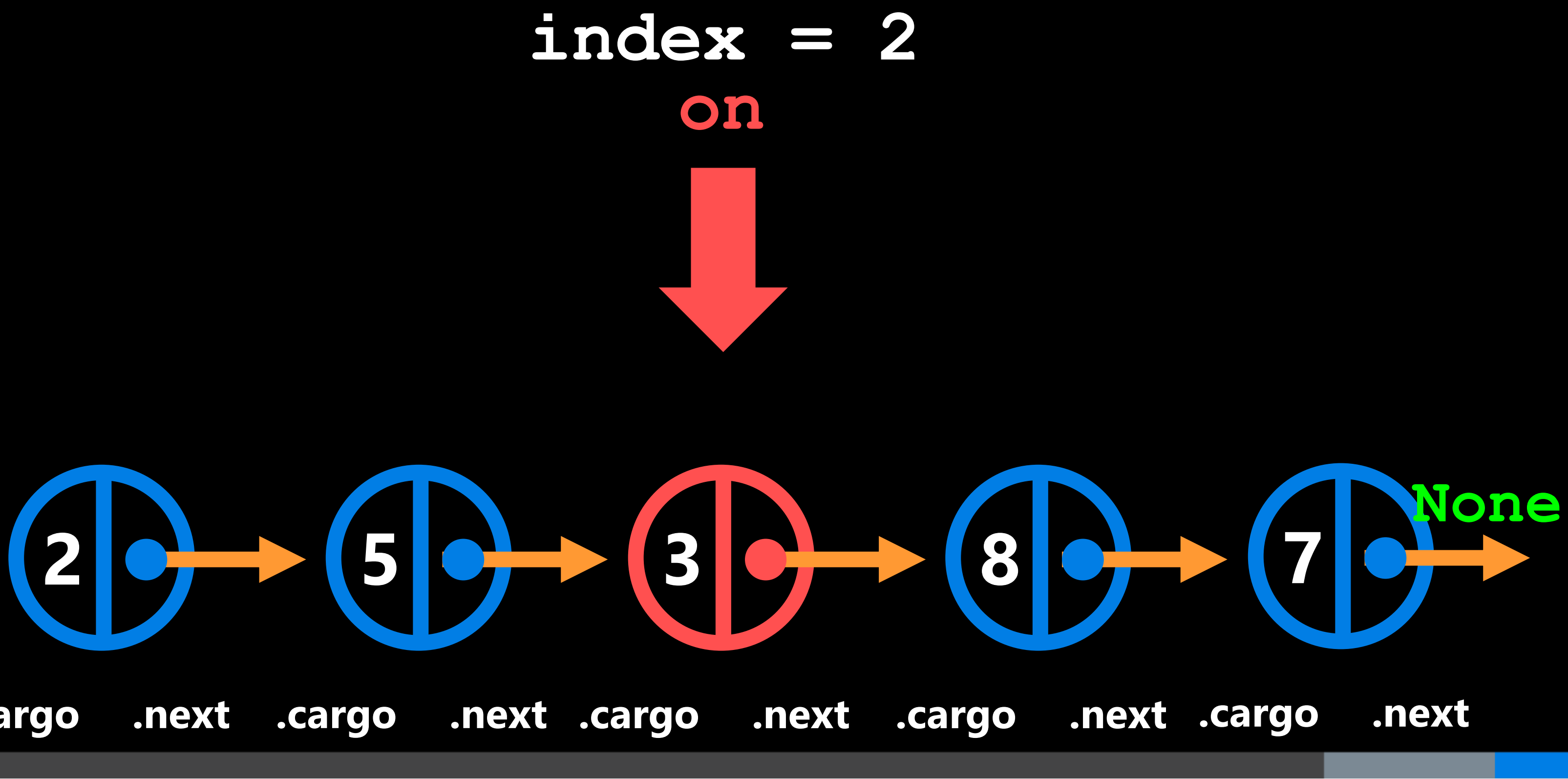
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

True

Move on to next position.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.get_at_index(3)
8
```

Get node at index = 3.



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

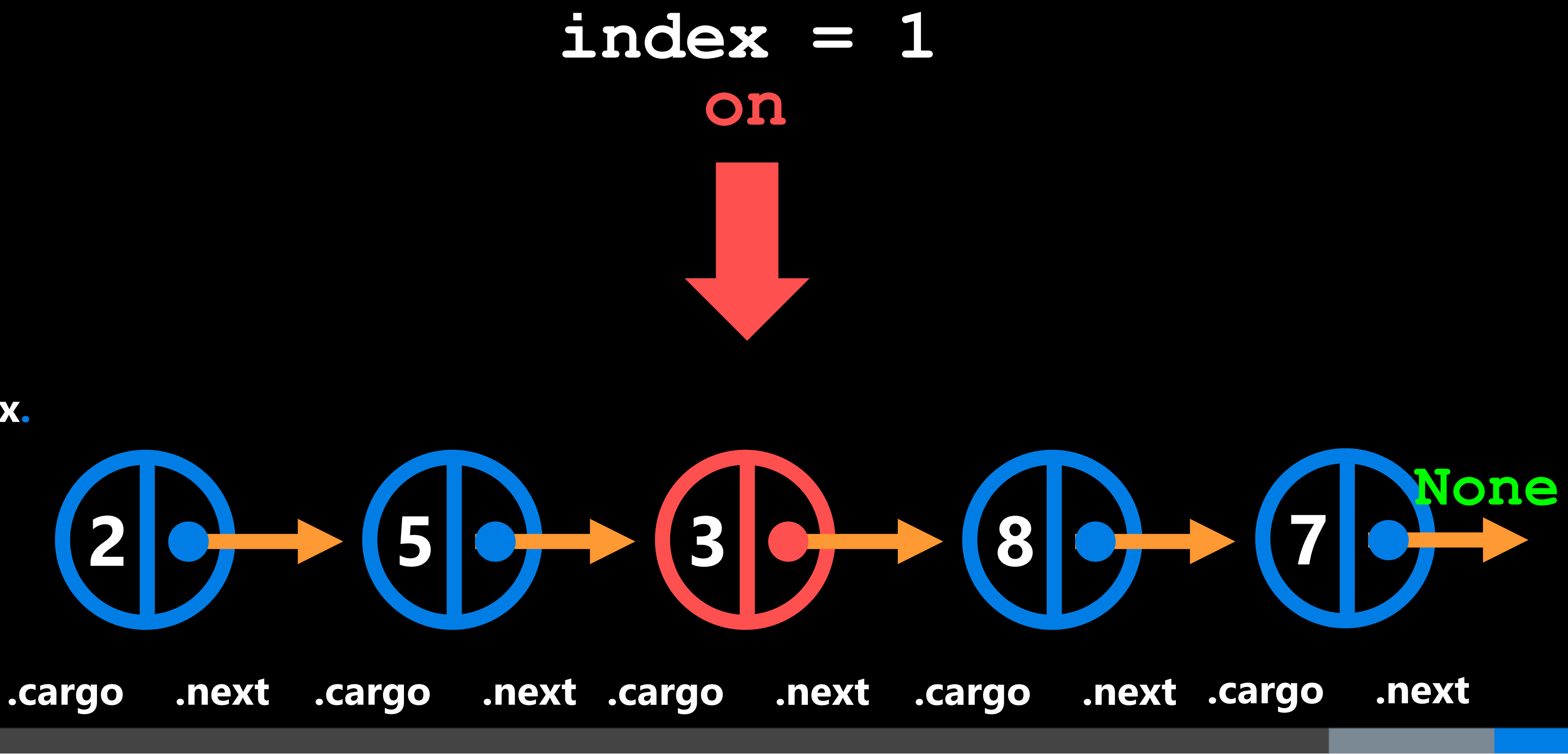
True → while on is not None and index != 0:

Update index. ← index -= 1

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. →

```
>>> linked_list.get_at_index(3)
8
```



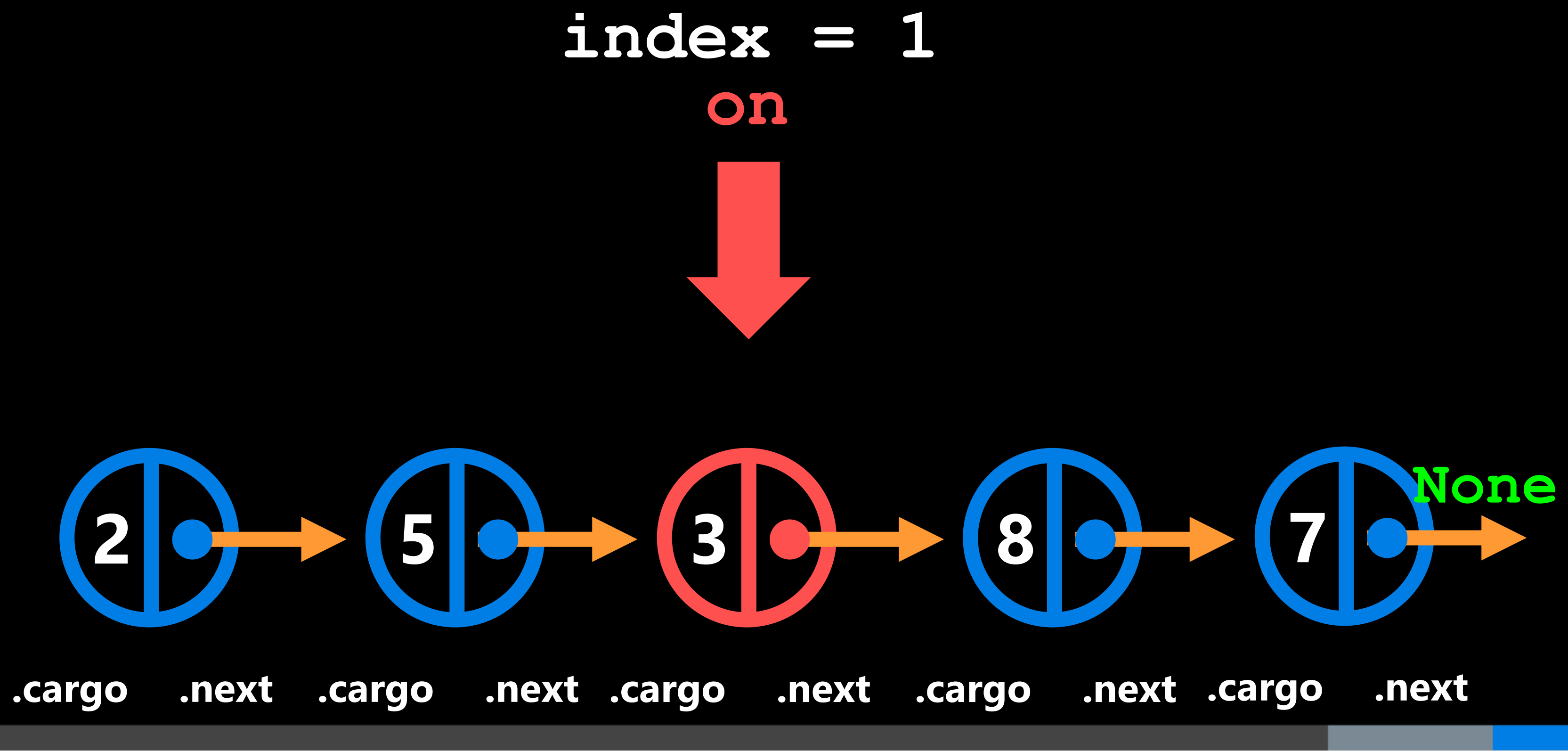
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

True ➡ while on is not None and index != 0:

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. ➡

```
>>> linked_list.get_at_index(3)
8
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

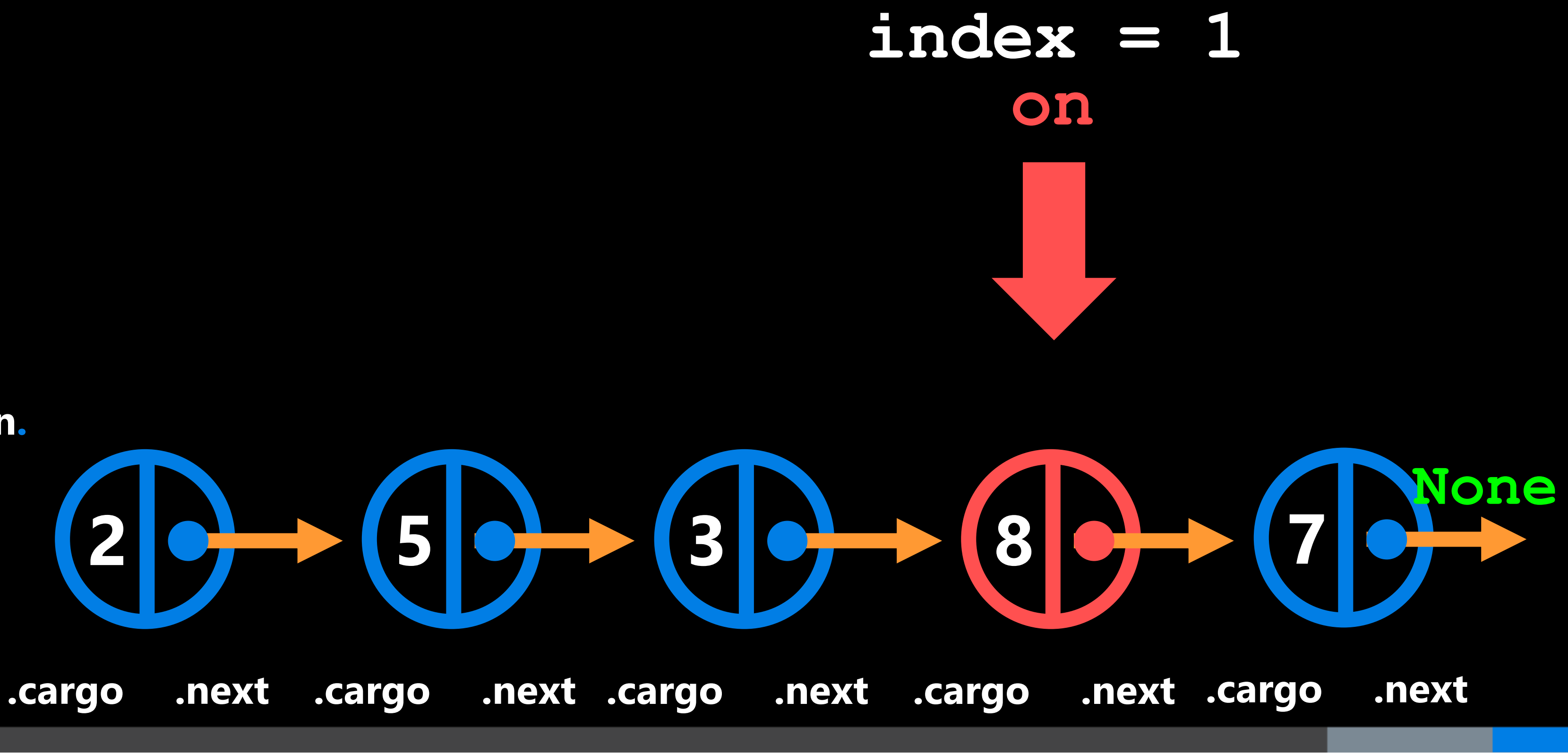
True → while on is not None and index != 0:

on = on.next ← **Move on to next position.**

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. →

```
>>> linked_list.get_at_index(3)
8
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

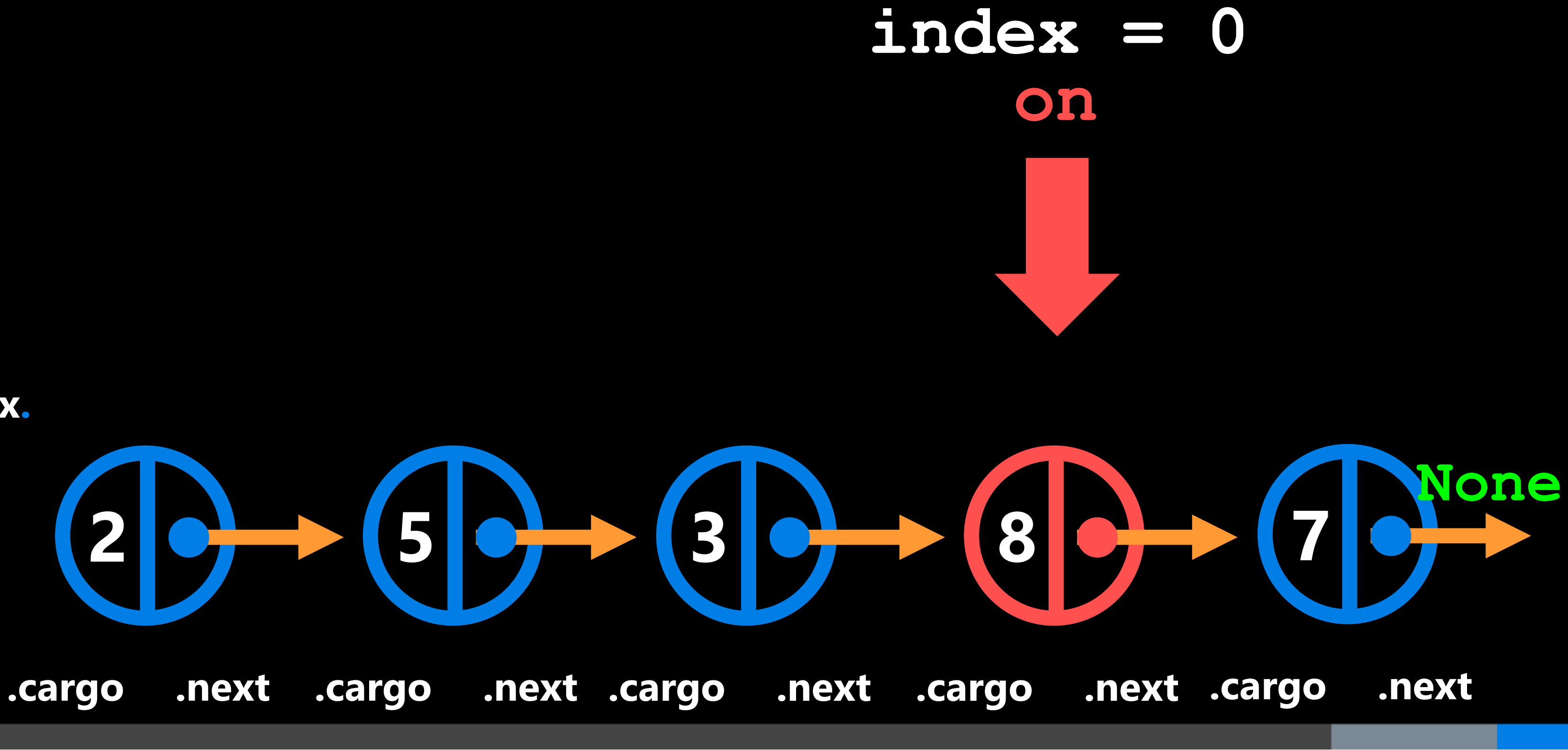
True

Update index.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3.

```
>>> linked_list.get_at_index(3)
8
```

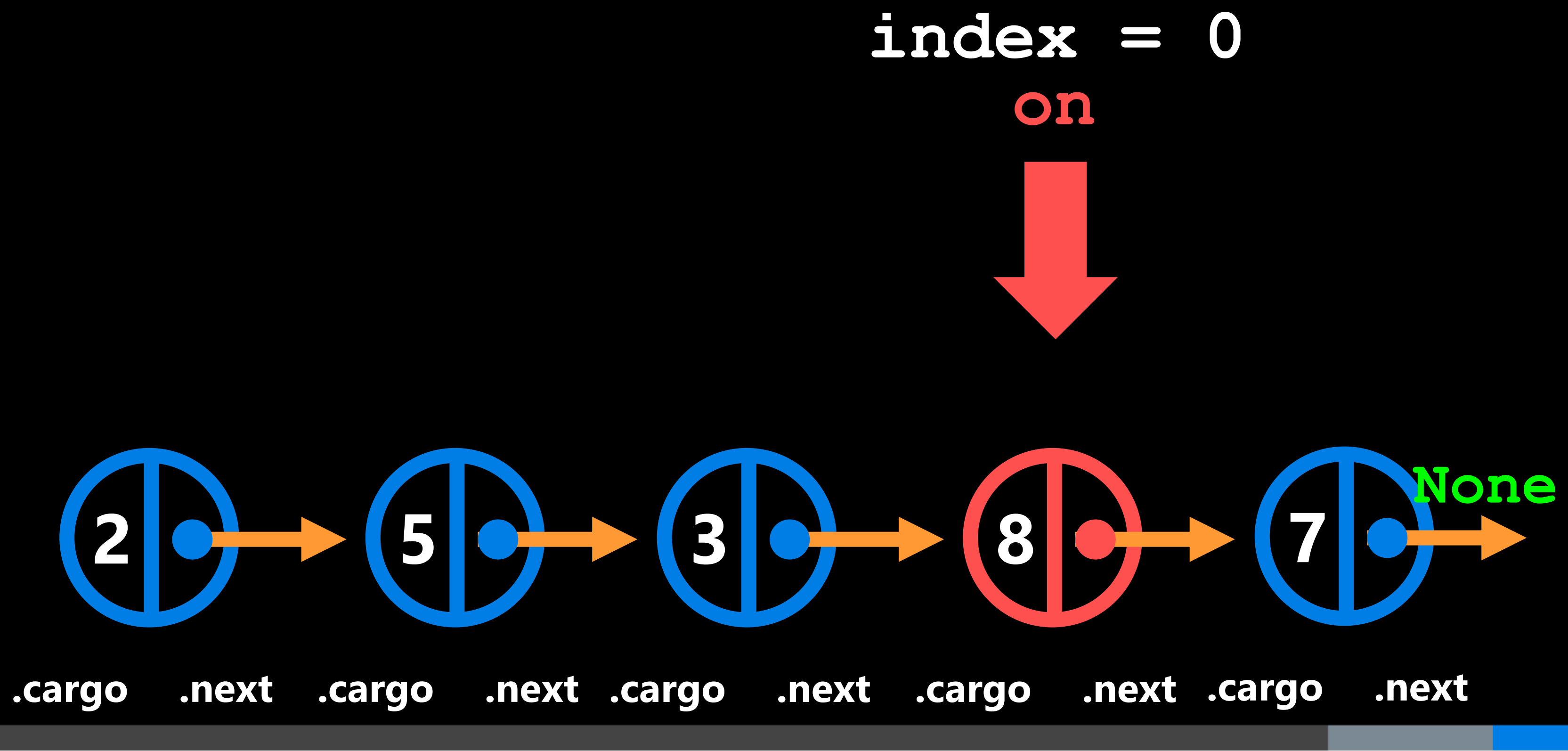


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

False → while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3. →

```
>>> linked_list.get_at_index(3)
8
```



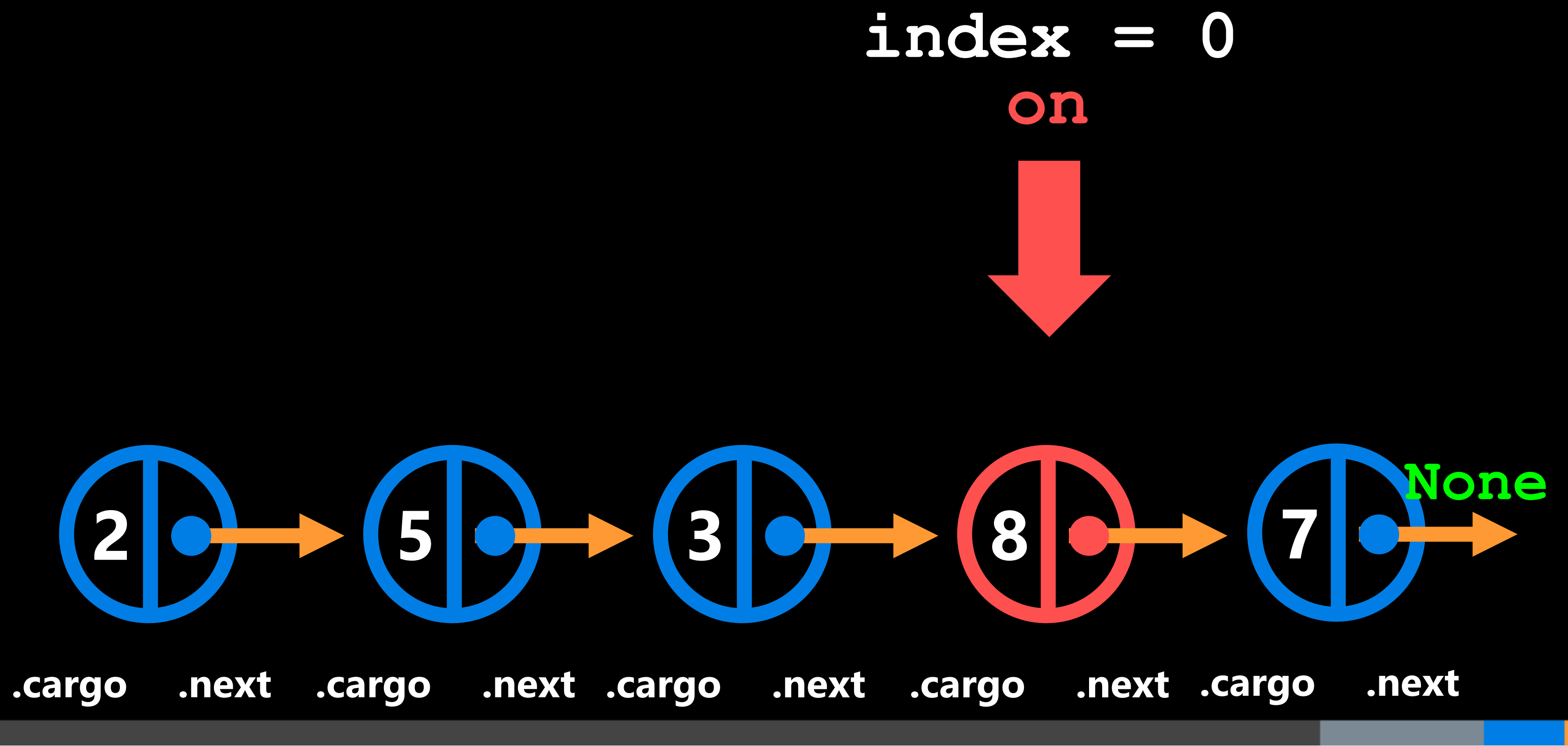
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

True

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3.

```
>>> linked_list.get_at_index(3)
8
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Add a new node at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo): ...
```

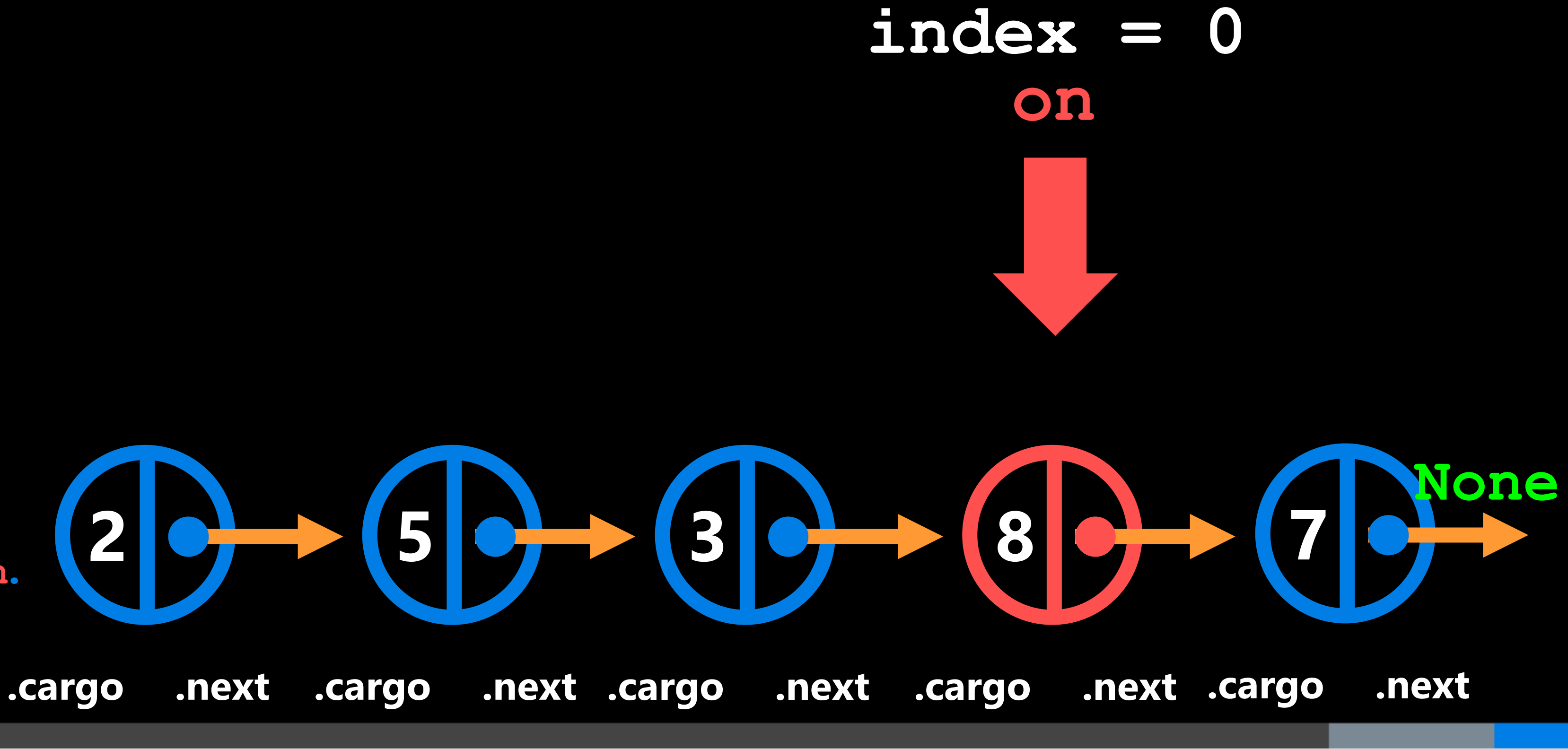
True

Return cargo at on.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```

Get node at index = 3.

```
>>> linked_list.get_at_index(3)
8
```

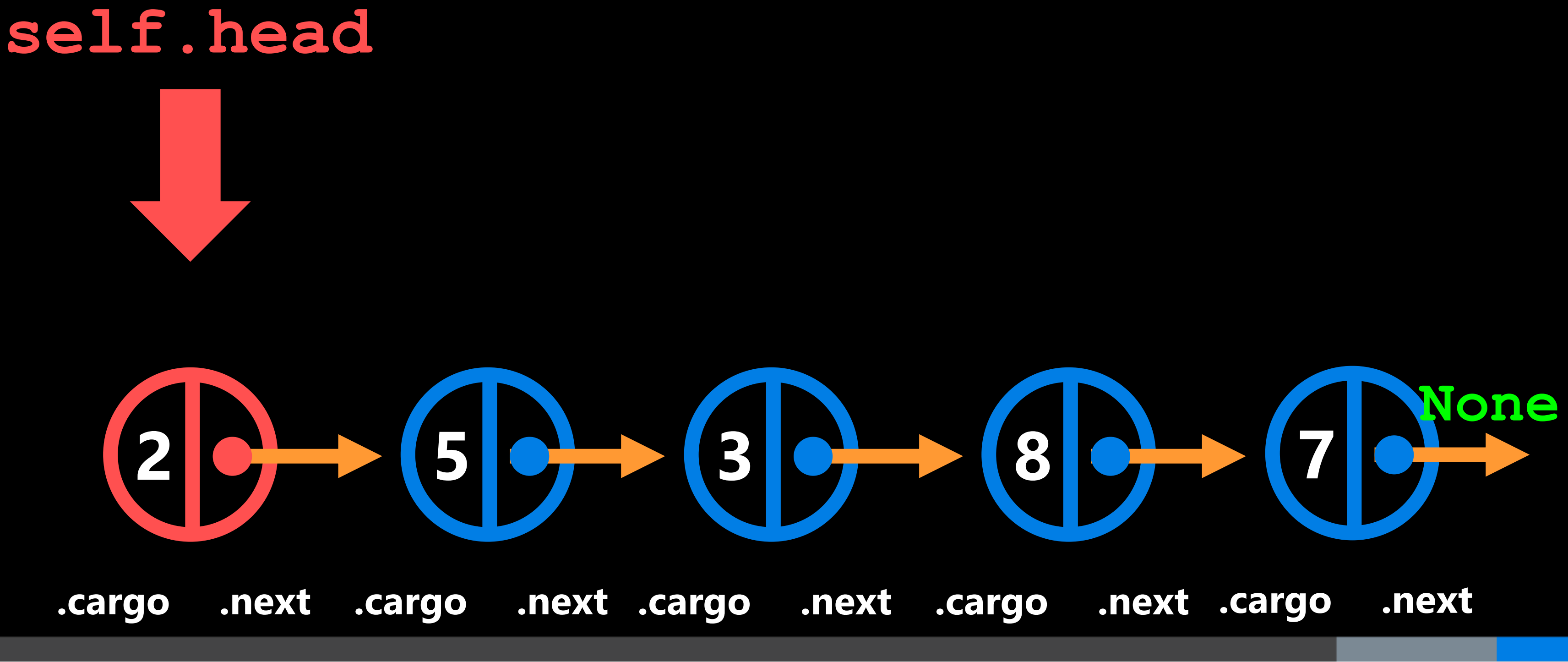


```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

delete_by_cargo method.

```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

        while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

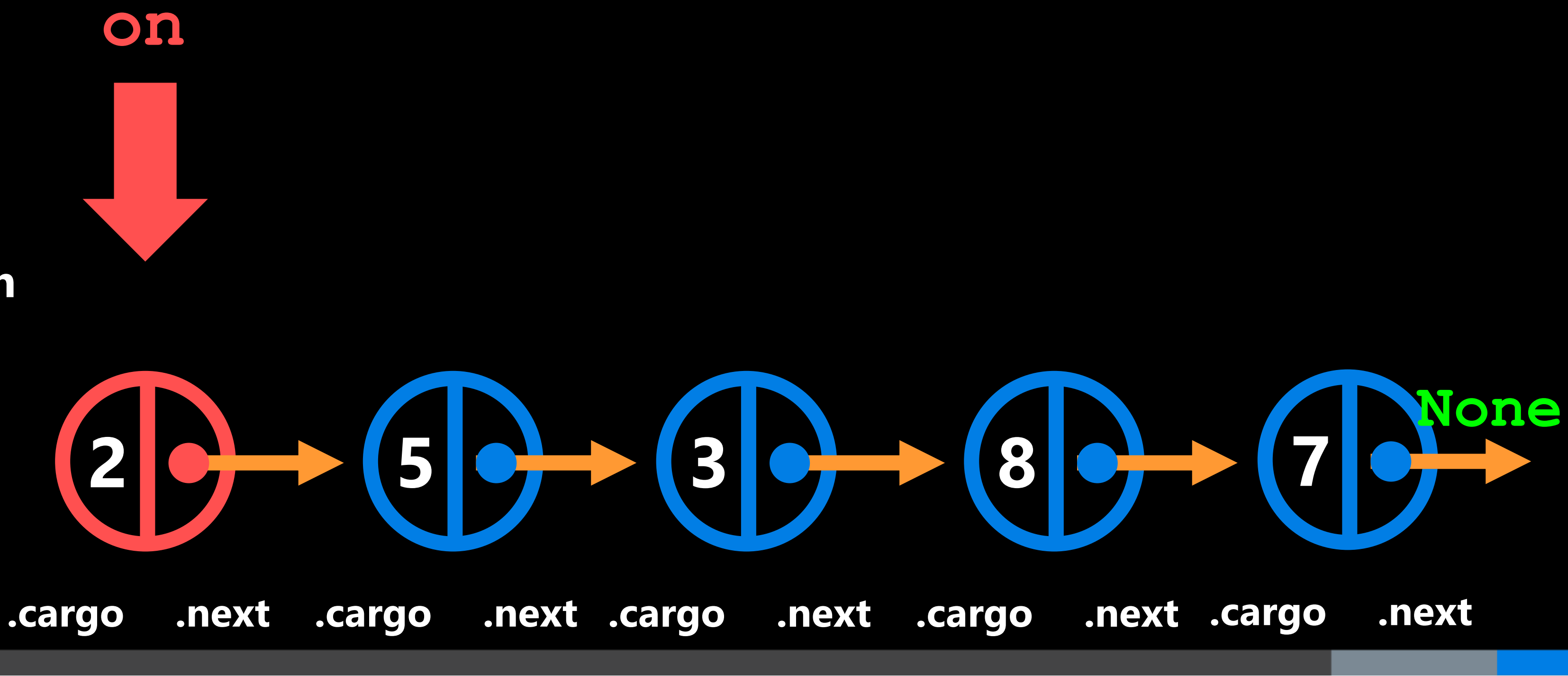
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

        while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

Set **on** position

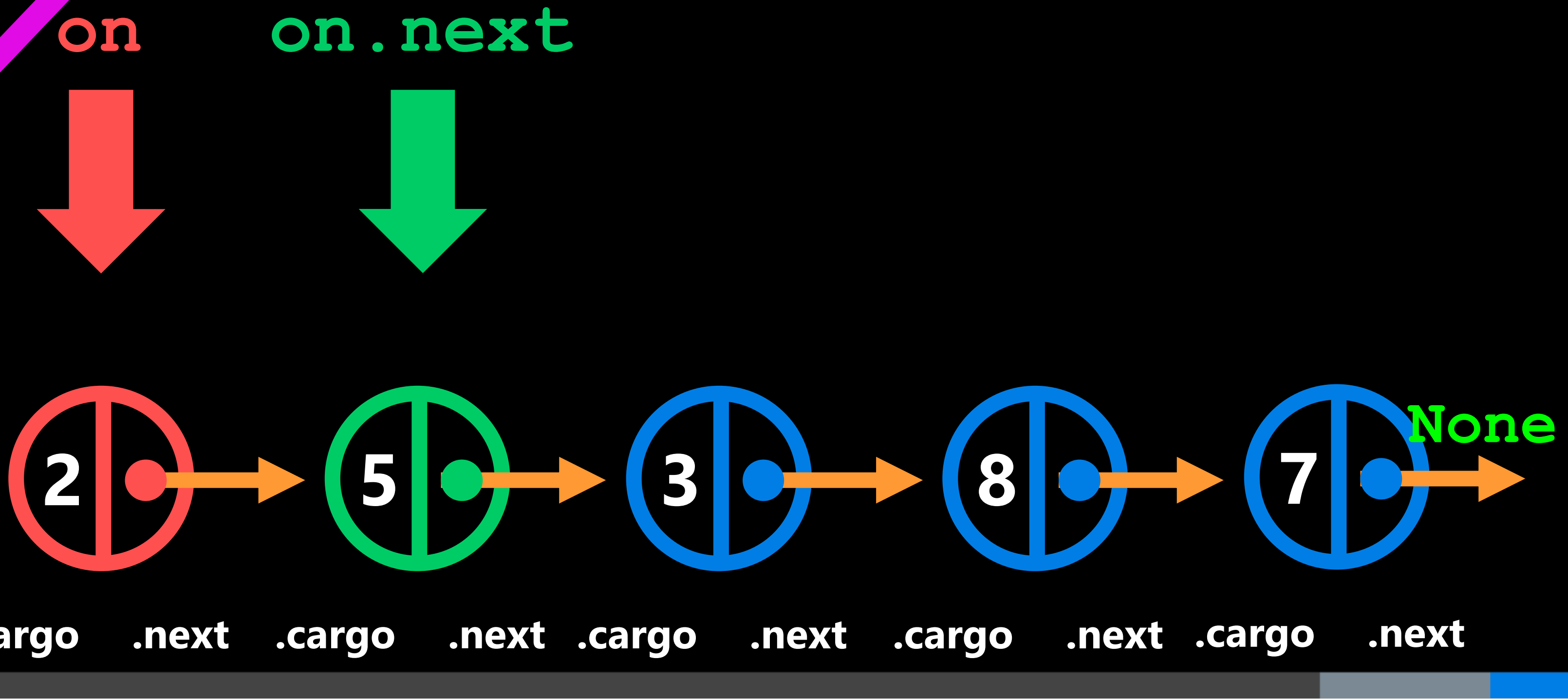
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

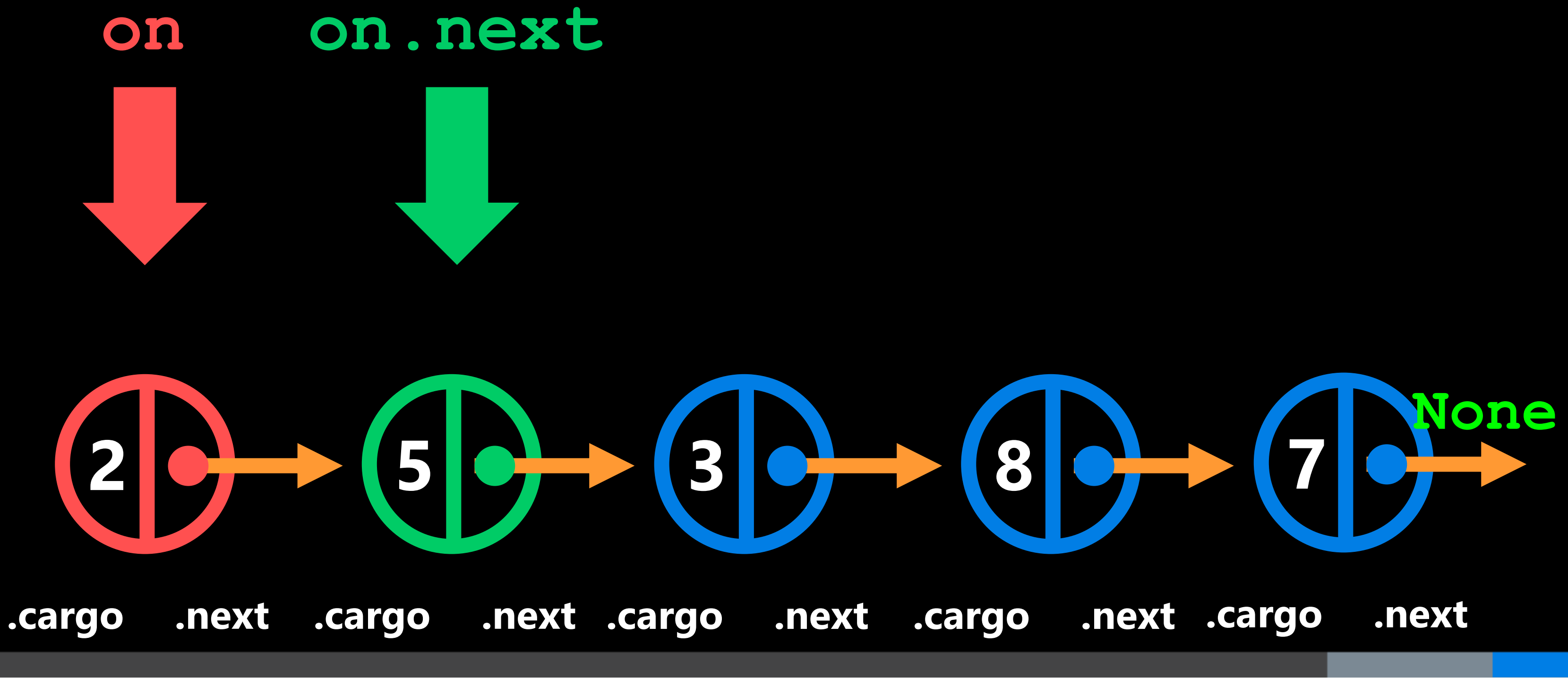
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```

`while on is not None and on.next is not None`



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

False ➡     if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



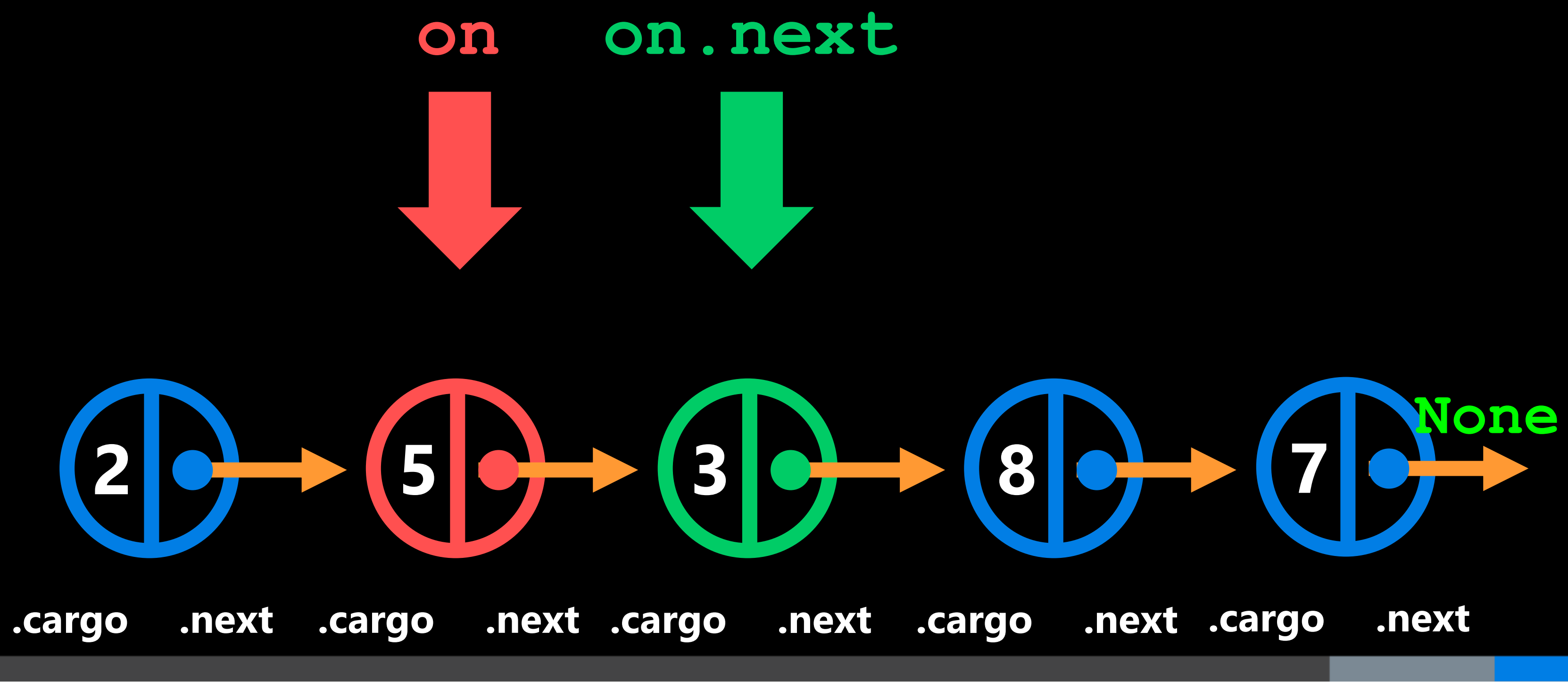
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

        while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

True

False

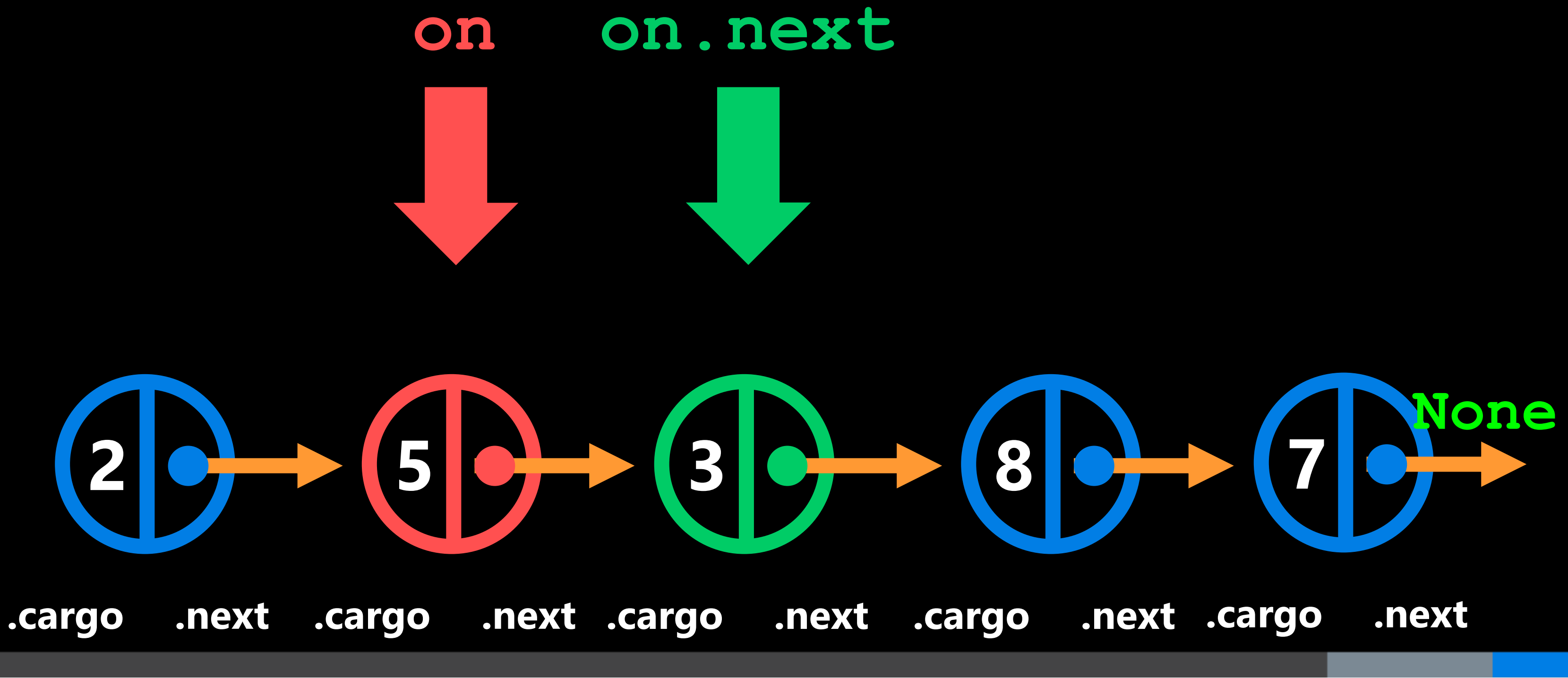
Move on to next position.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



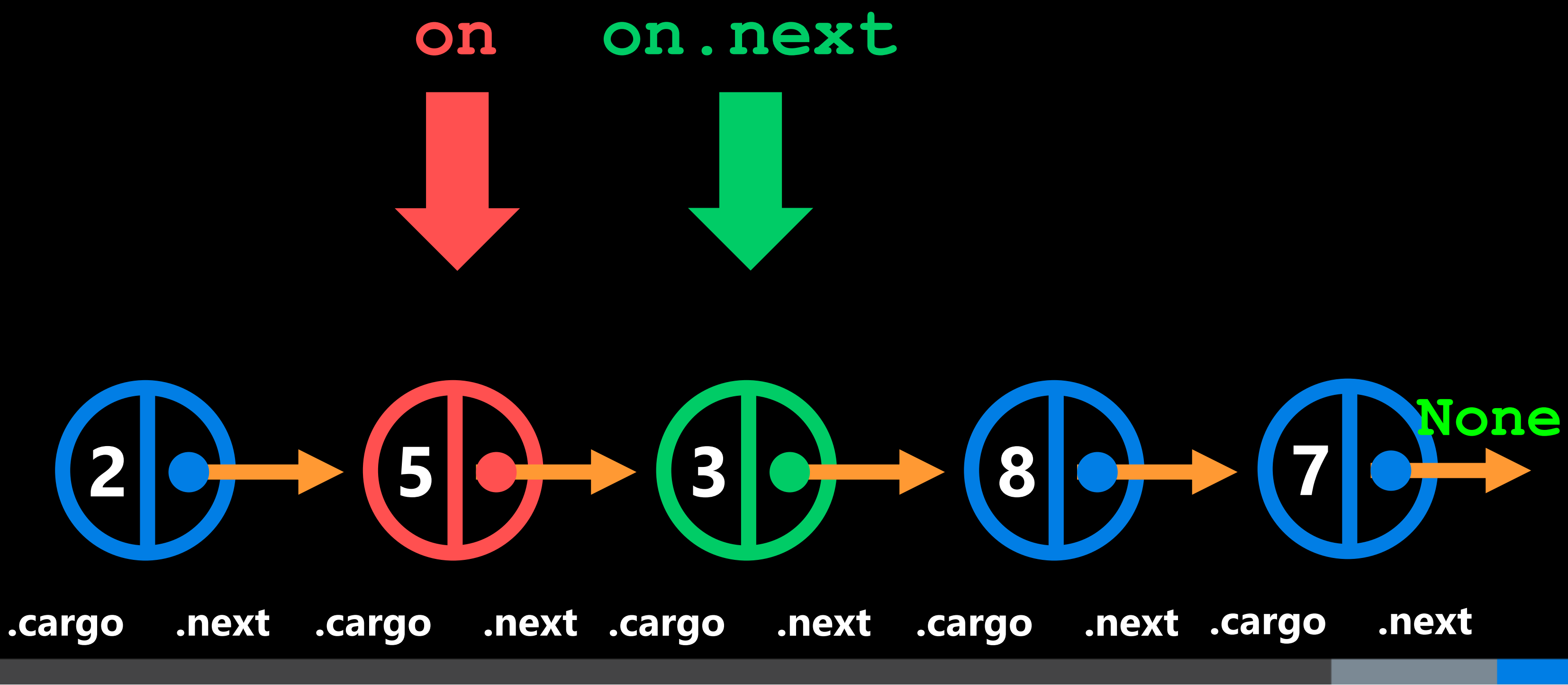
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

    True ➡  if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



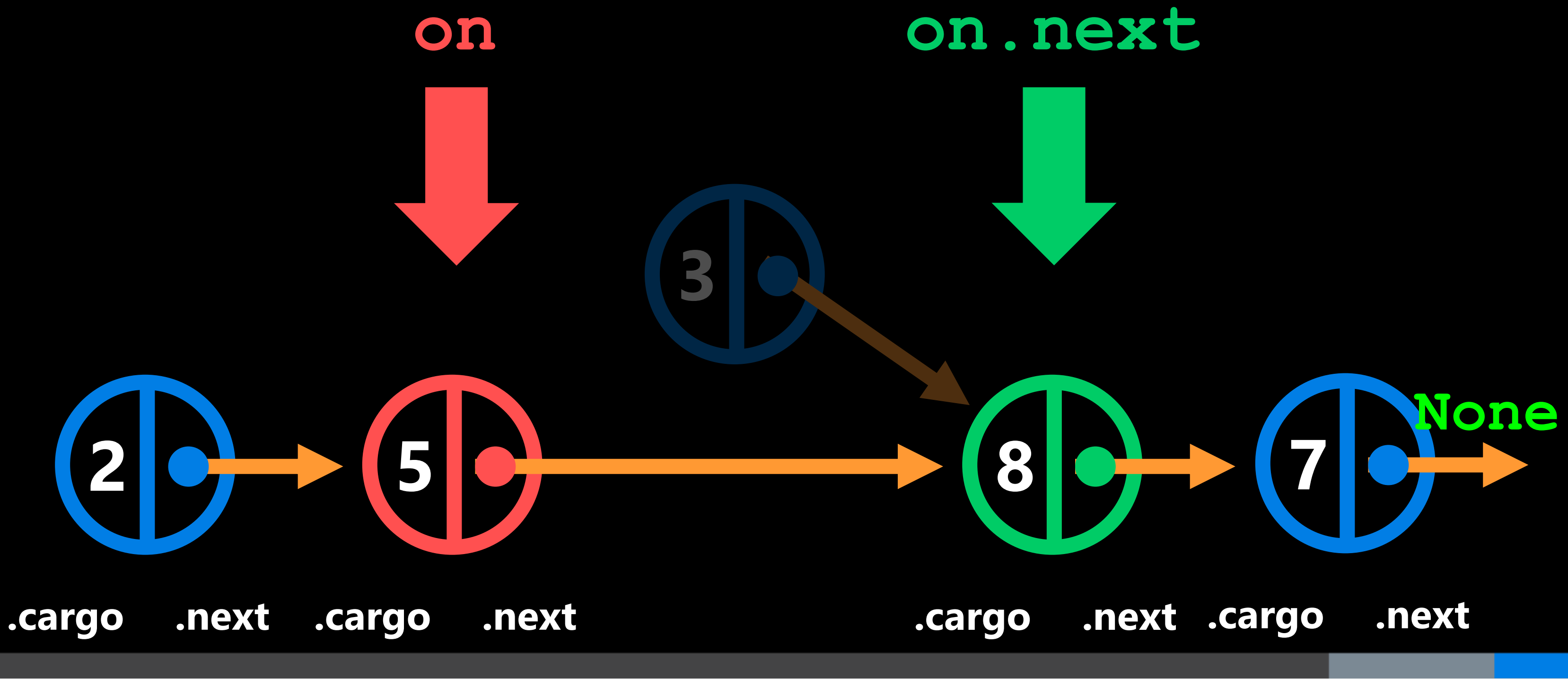
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

        while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

True

True

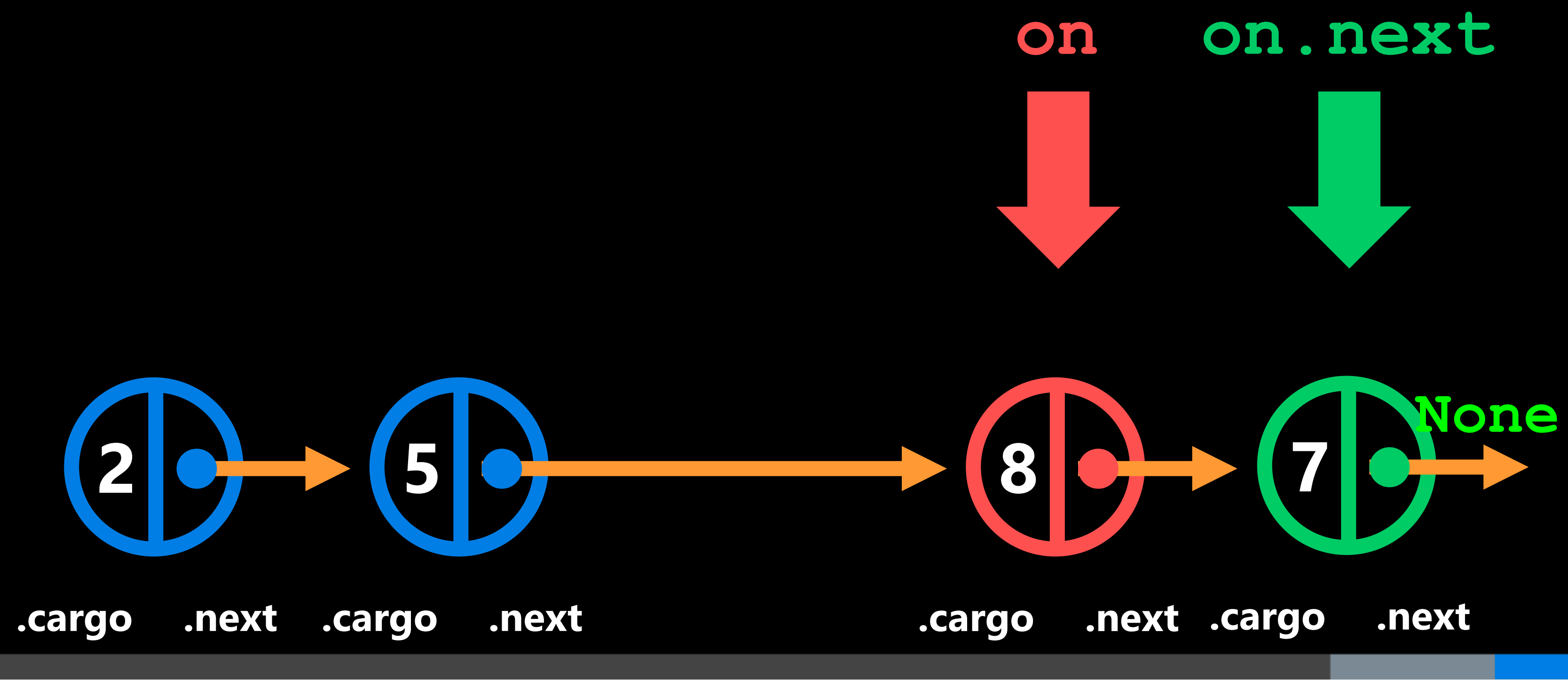
Update pointer.

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



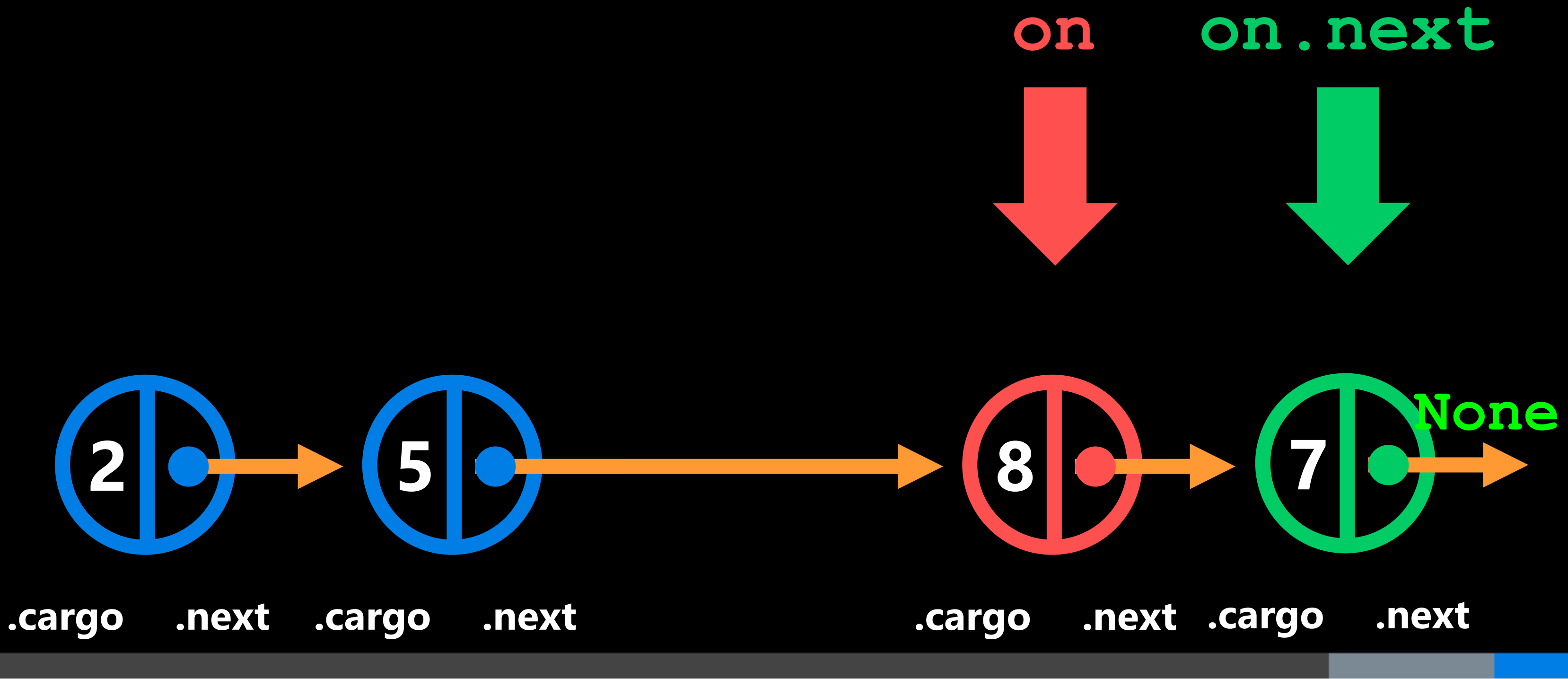
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

    True ➡  if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next ⬅ Move on to
                           next position.
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



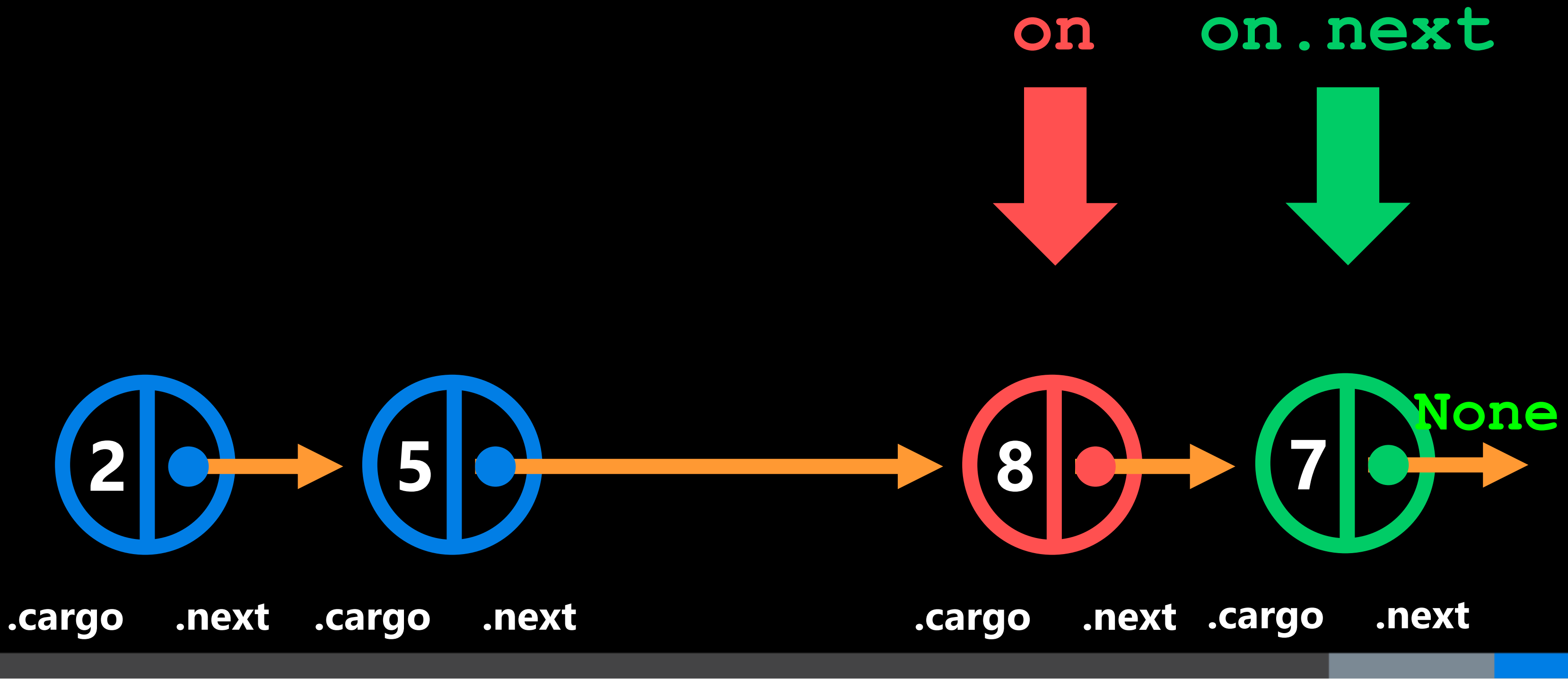
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

True ➡  while on and on.next:

 False ➡    if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



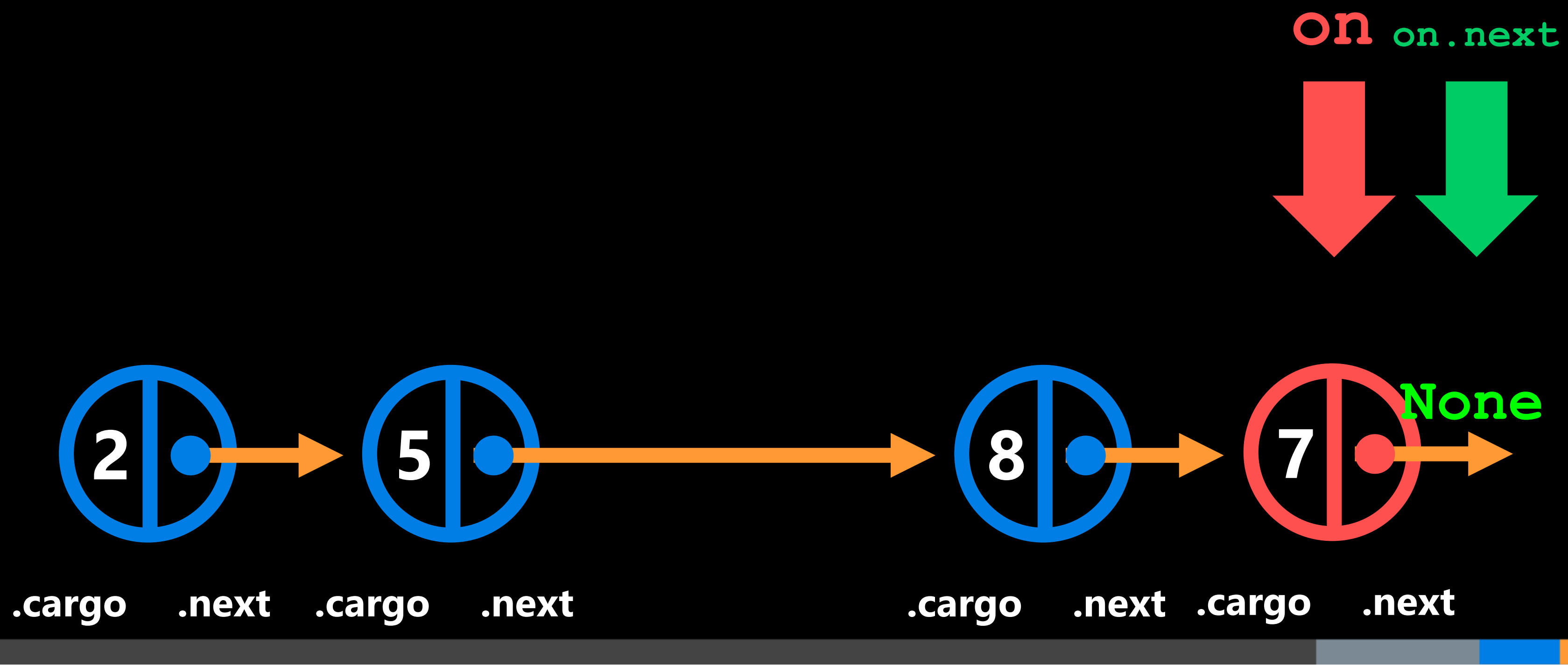
```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

        while on and on.next:
            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

True ➡ while on and on.next:

False ➡ if on.next.cargo == cargo:

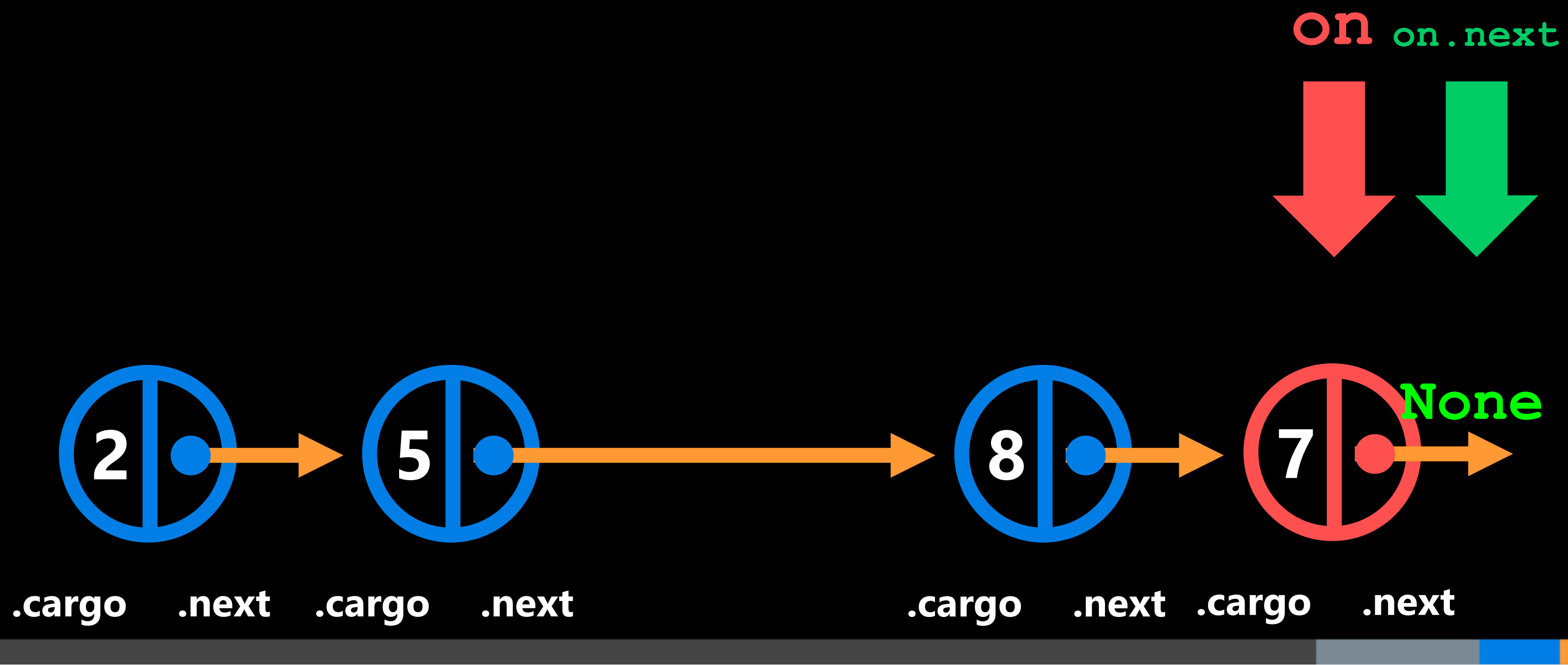
on = on.next ⬅ **Move on to next position.**

```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



```
class LinkedList:
    """A class that implements a linked list."""

    def __init__(self):
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo): ...

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain
        cargo value.
        """
        on = self.head

False ➡ while on and on.next:

            if on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
```

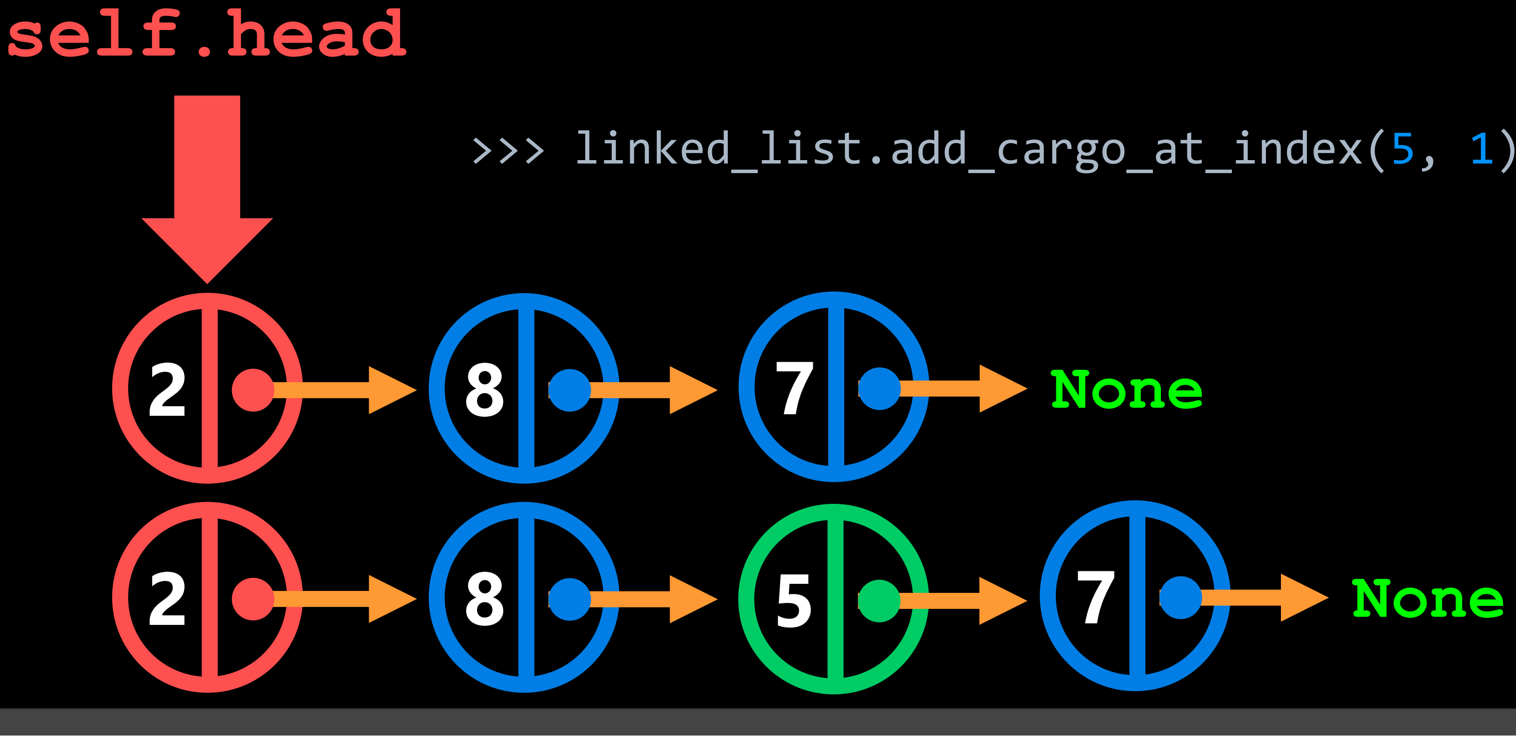
```
>>> linked_list = LinkedList()
>>> linked_list.add_to_head(7)
>>> linked_list.add_to_head(8)
>>> linked_list.add_to_head(3)
>>> linked_list.add_to_head(6)
>>> linked_list.add_to_head(2)
>>> linked_list.__str__()
'(2) --> (5) --> (3) --> (8) --> (7) --> None'

>>> linked_list.delete_by_cargo(3)
```



Breakout Session

- Let's create a new method to insert a new node after a specified index.

```
>>> linked_list.add_cargo_at_index(5, 1)
```



Open your notebook

Click Link:
3. Breakout Session

binary trees.

Node Based Data Structures

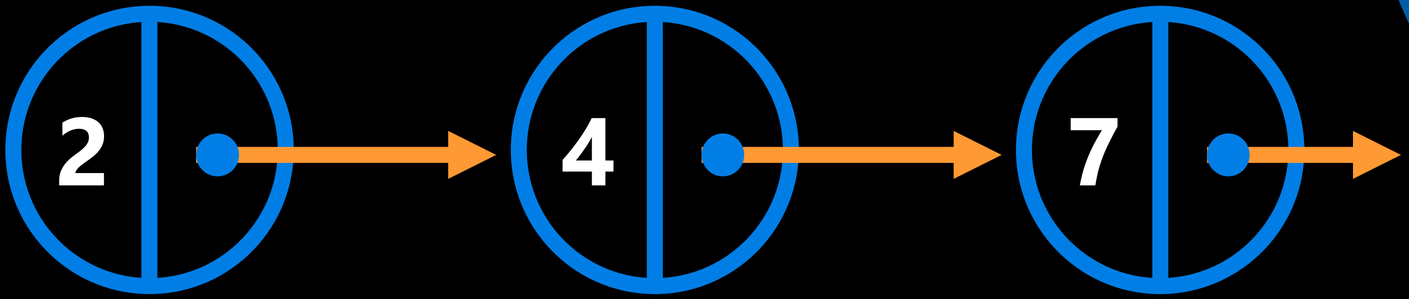
- Linked Lists and Binary trees are part of a family of node-based data structures.

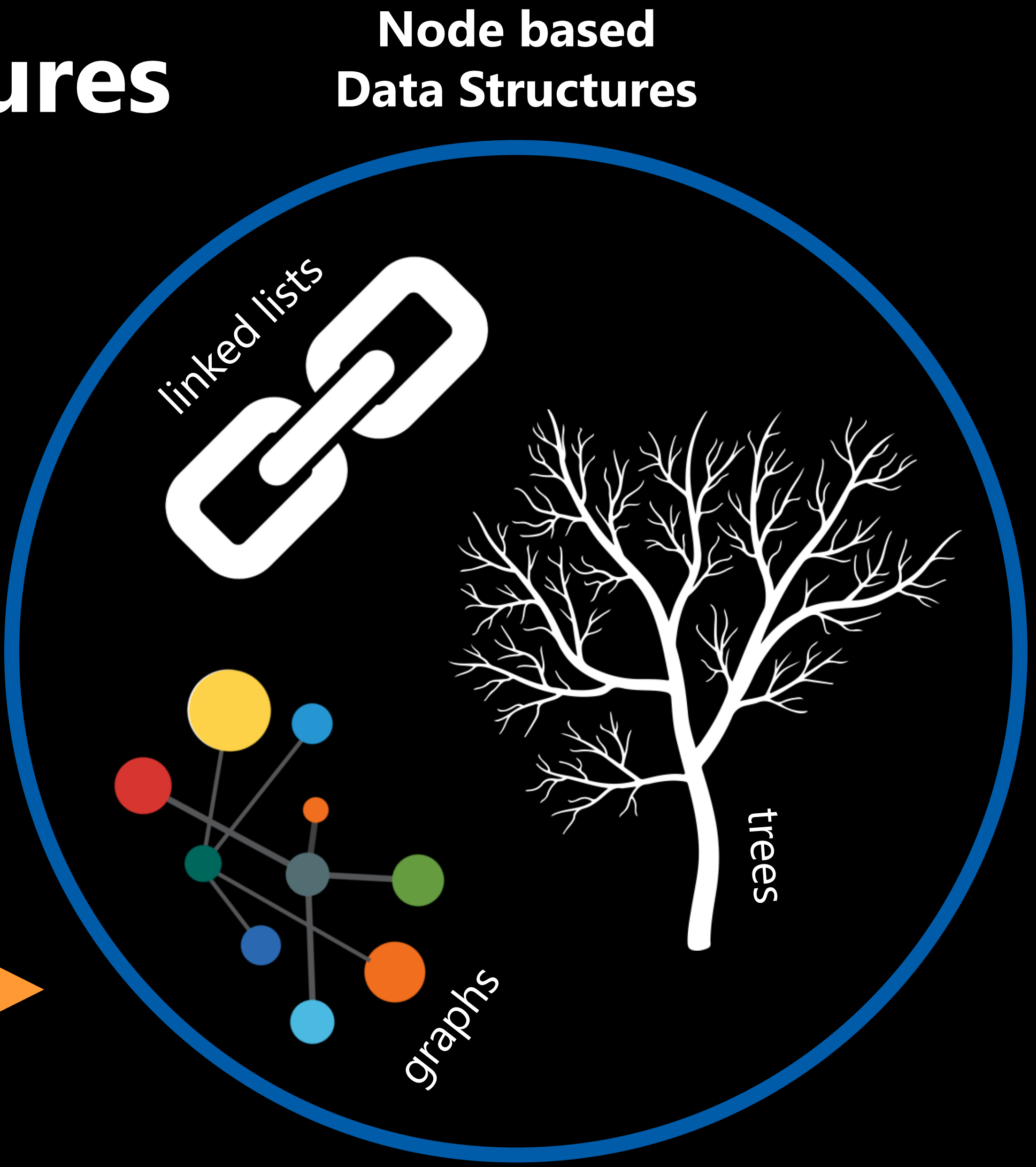
Node based
Data Structures

Node Based Data Structures

- You'll recall linked lists are made of up a series of nodes with a value property and a pointer.

Value
Pointer



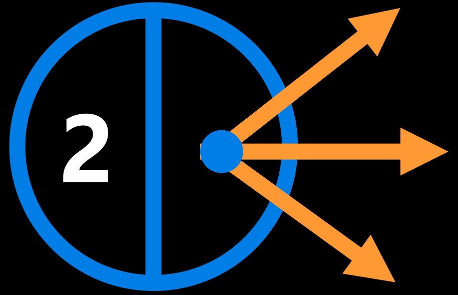


Node Based Data Structures

- Node based data structures are made up of **data** (value property) and **structure** (1 or more pointers).

Value

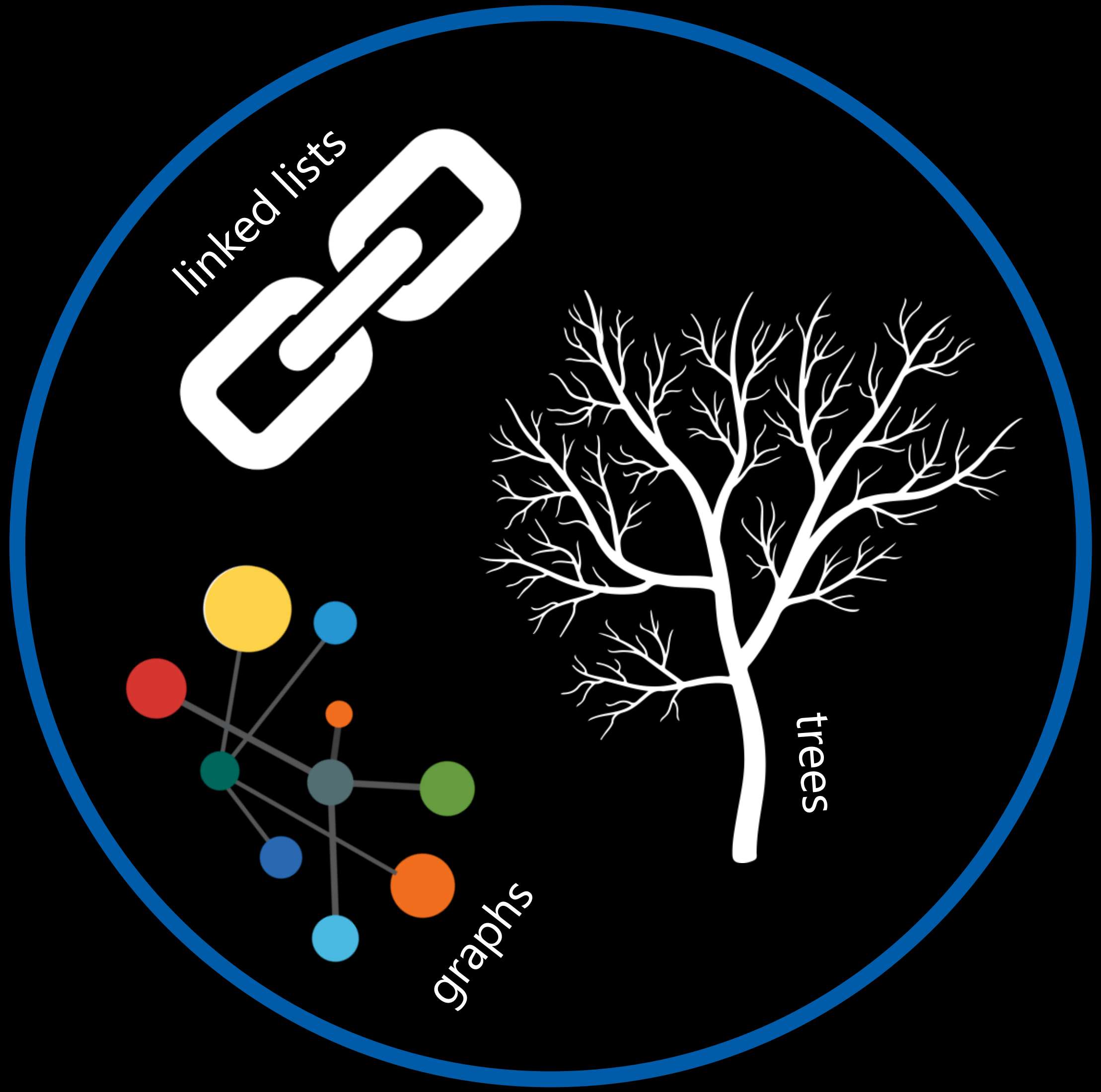
Pointer



(data)
value property

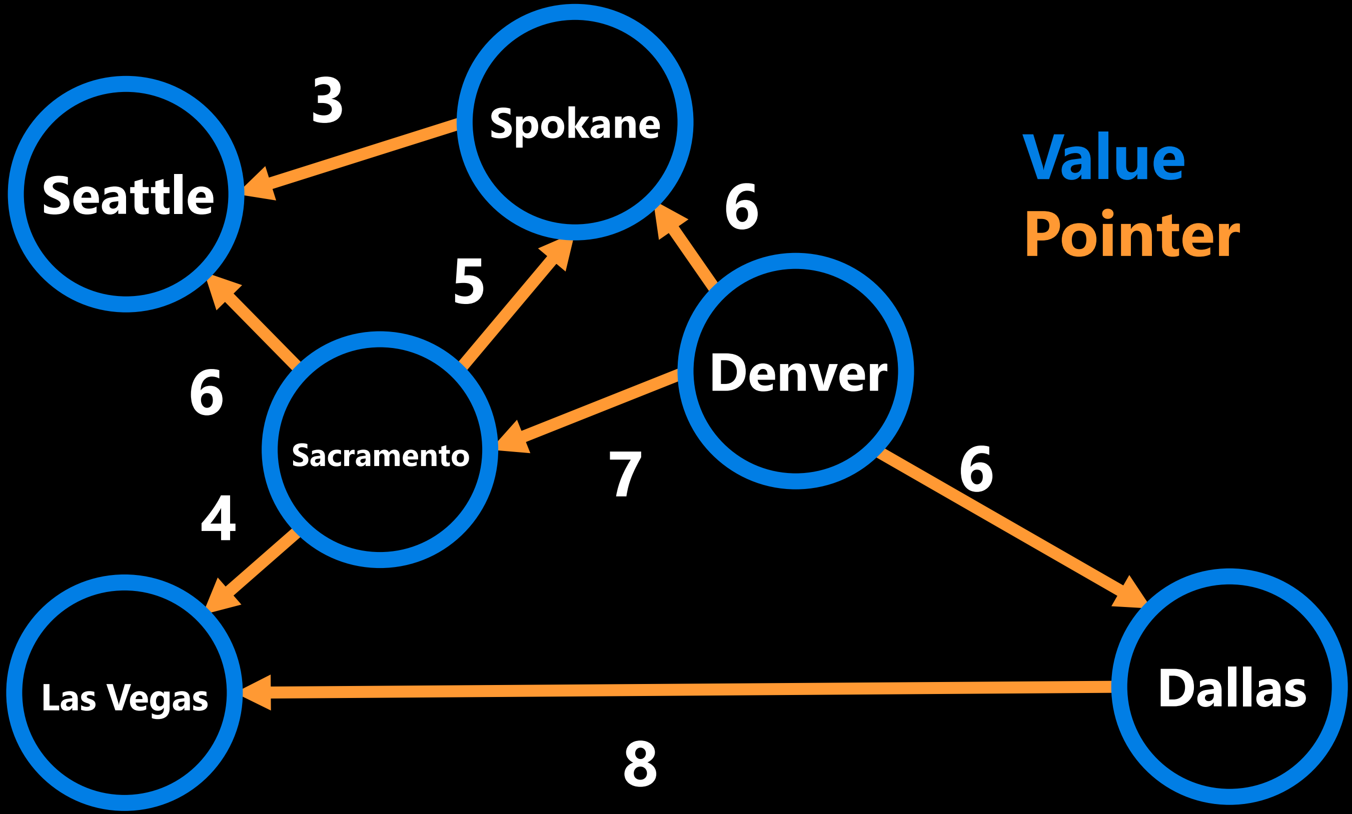
(structure)
1+ pointers

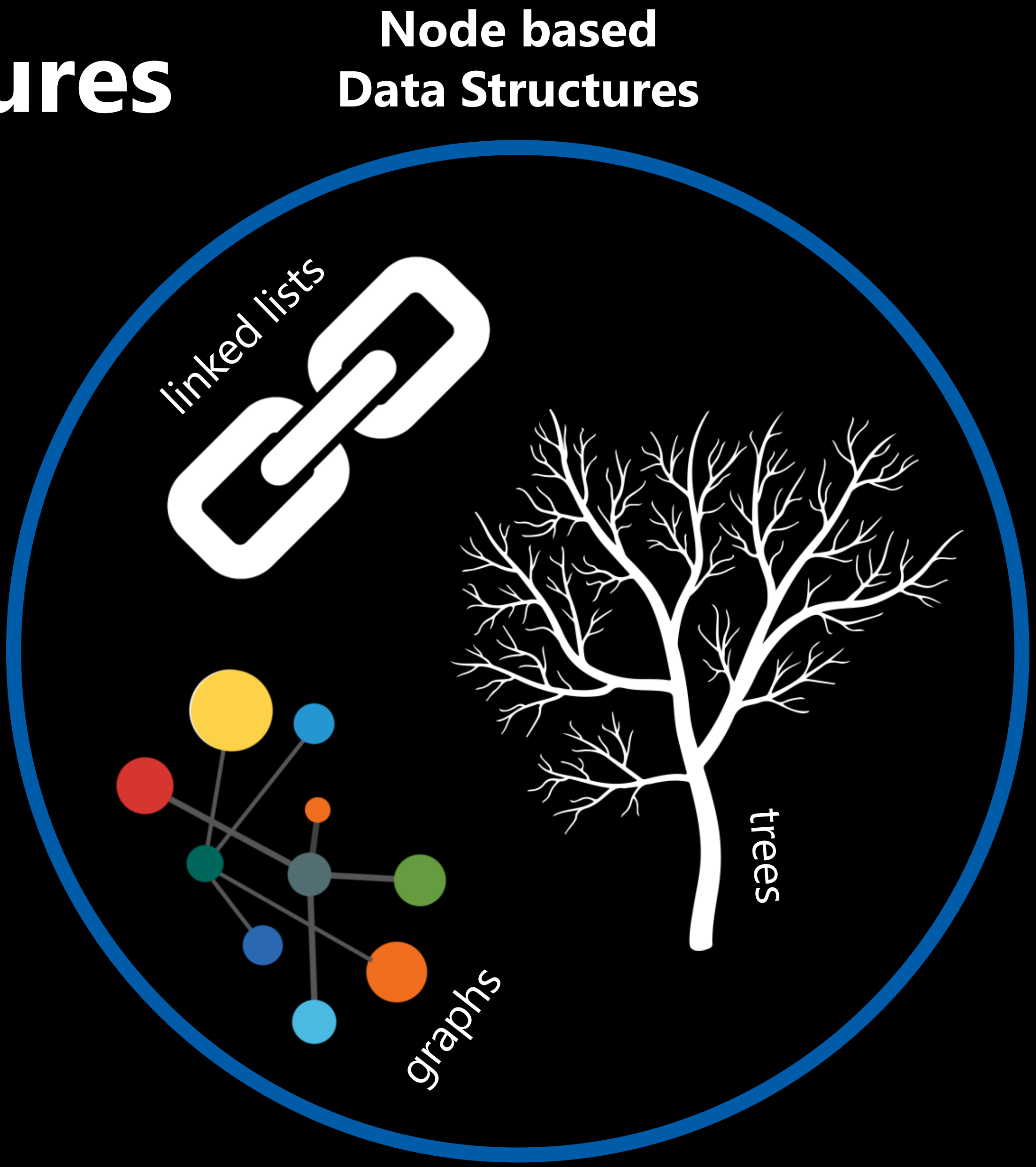
Node based
Data Structures



Node Based Data Structures

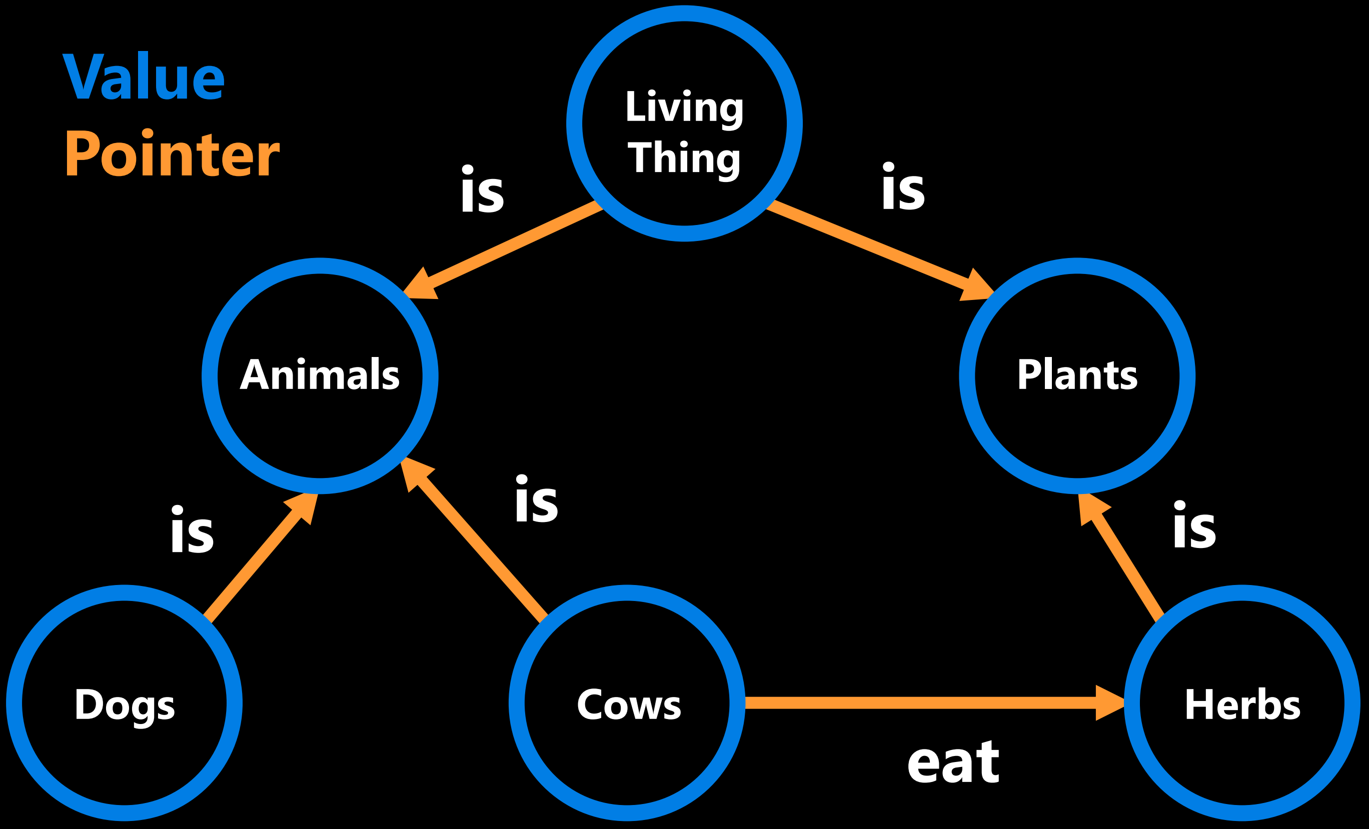
- Using the node data structure, we can create graphs like this.

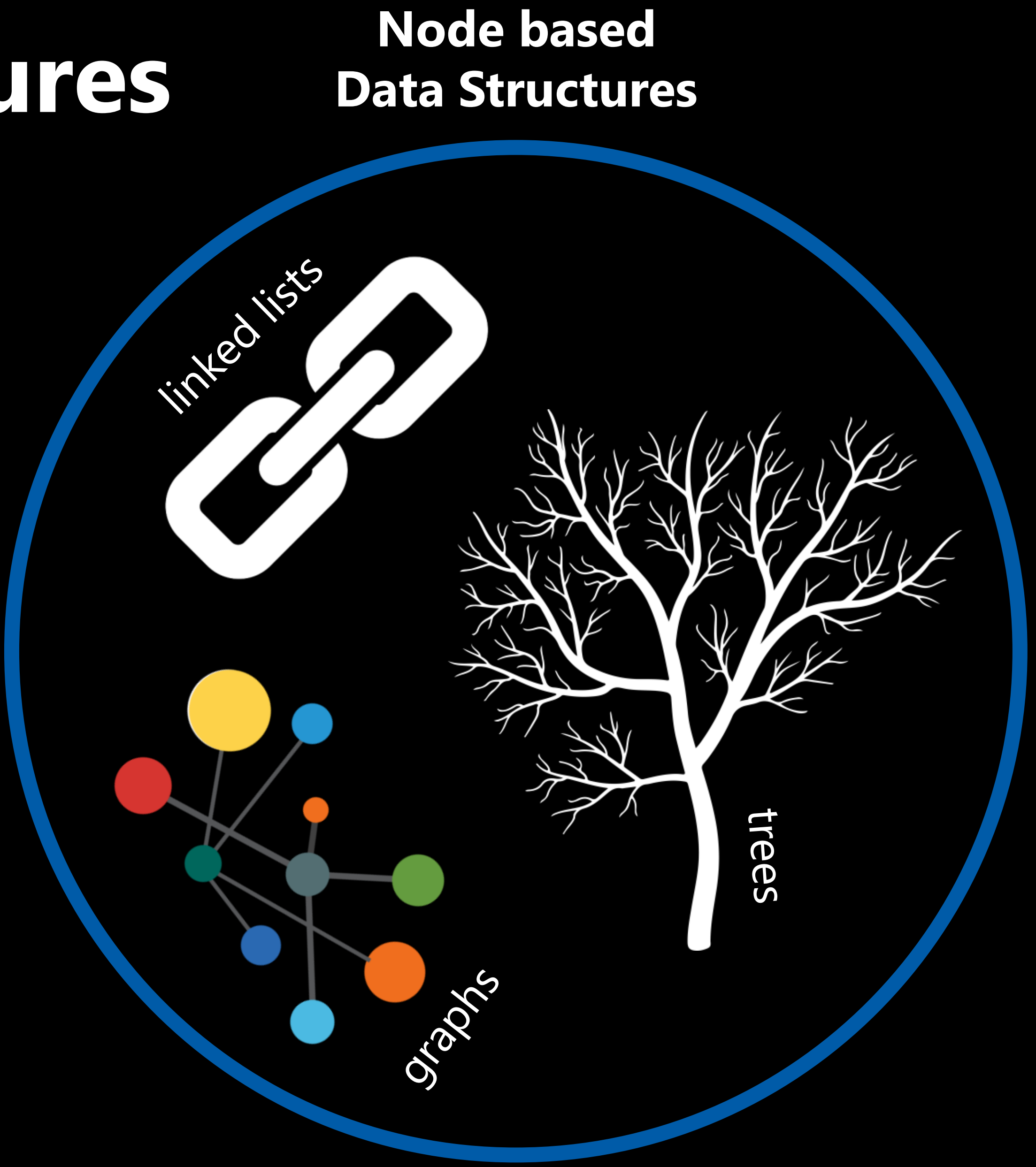




Node Based Data Structures

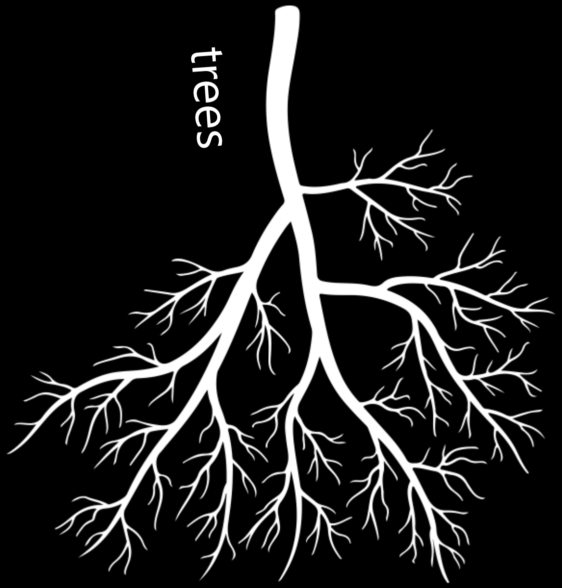
- Or a knowledge graph like this.

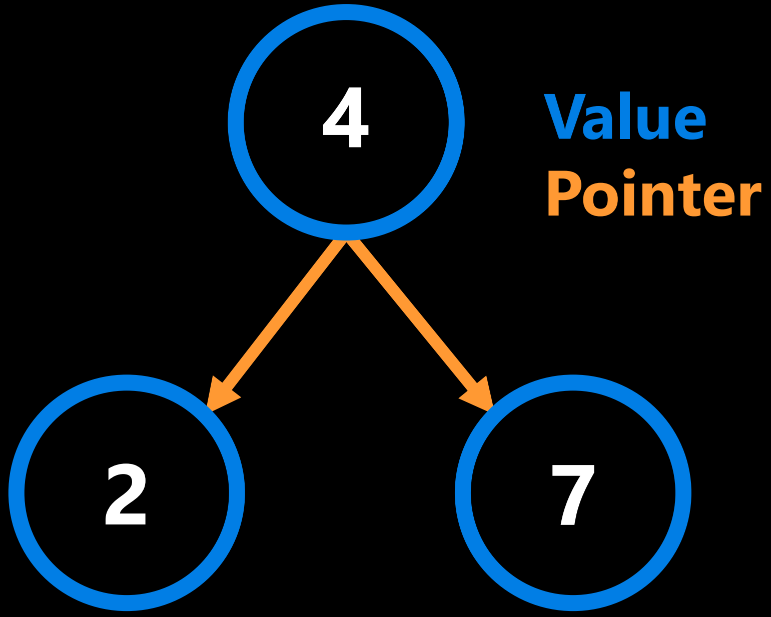




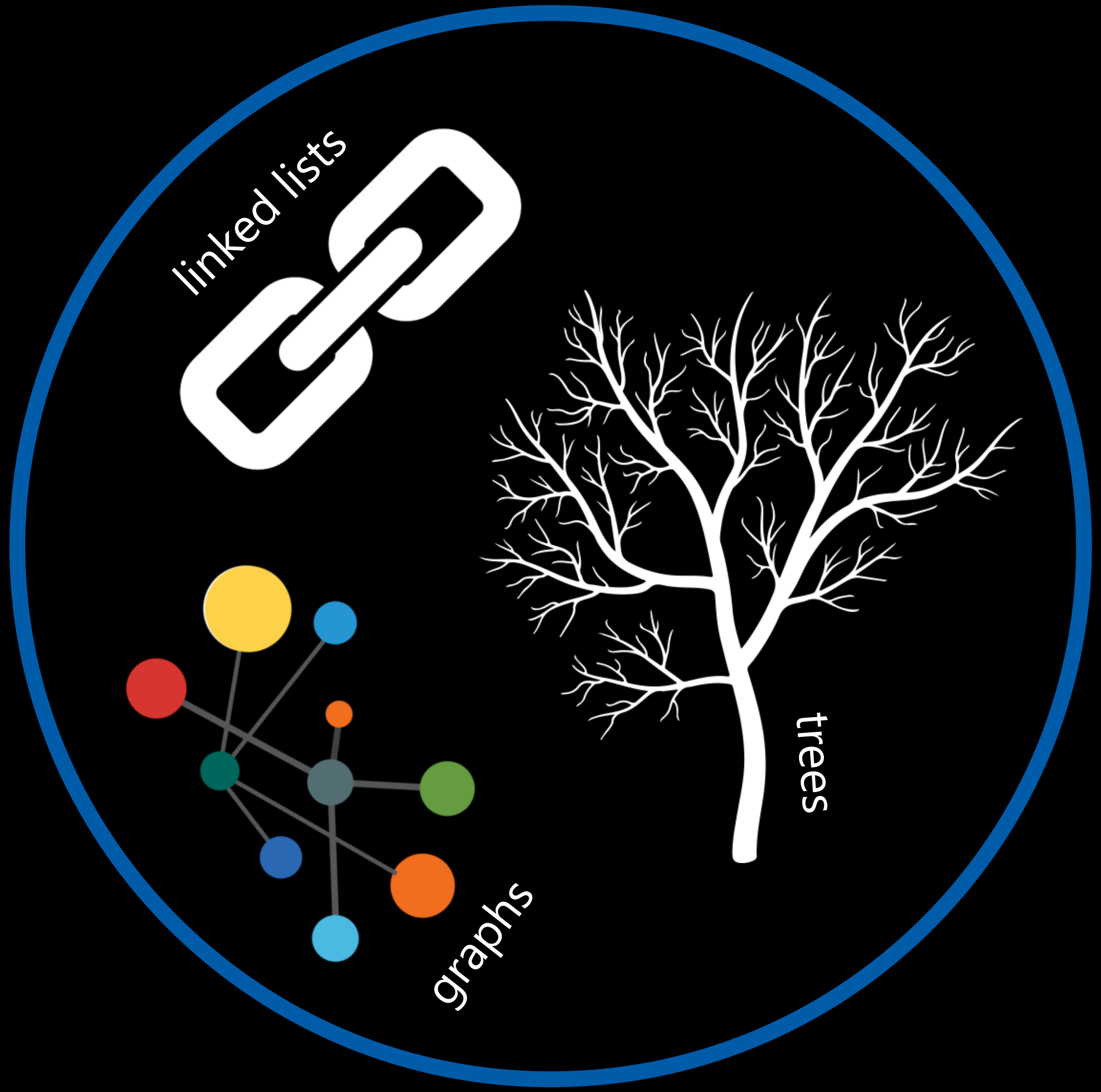
Node Based Data Structures

- Next, we have trees.
  - Trees are hierarchical with nodes branching in one direction with multiple pointers.



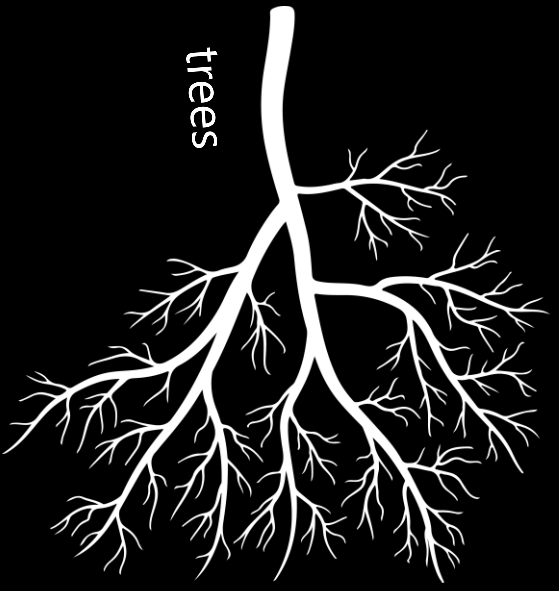


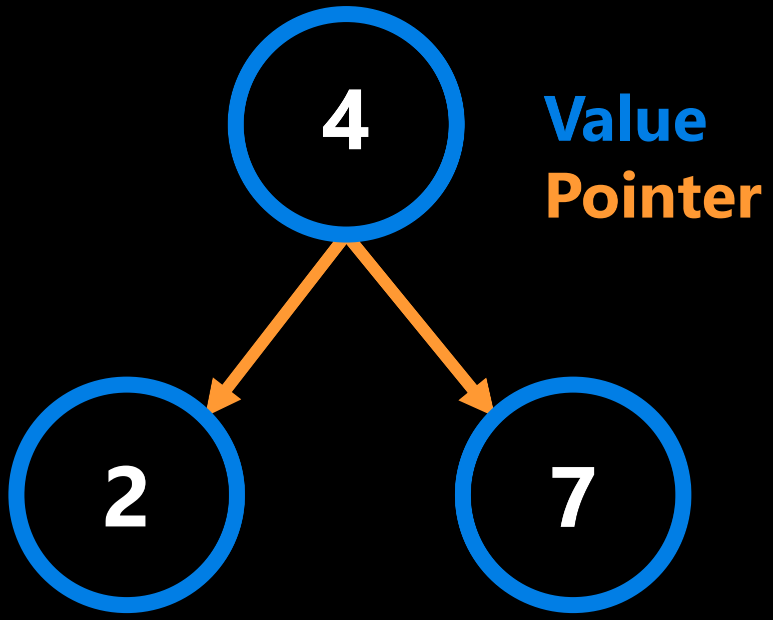
Node based Data Structures



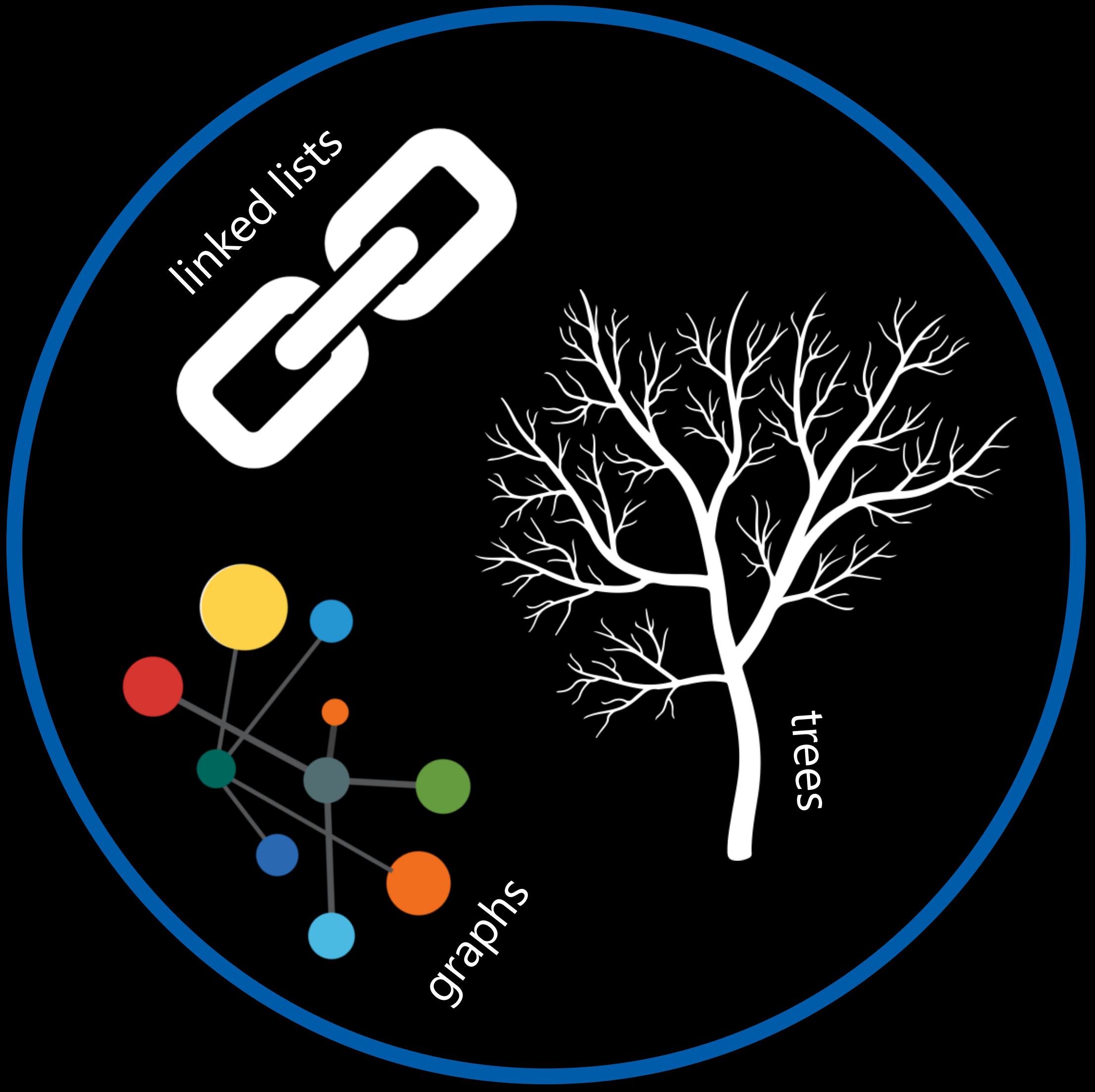
Node Based Data Structures

- Next, we have trees.
  - Trees are hierarchical with nodes branching in one direction with multiple pointers.



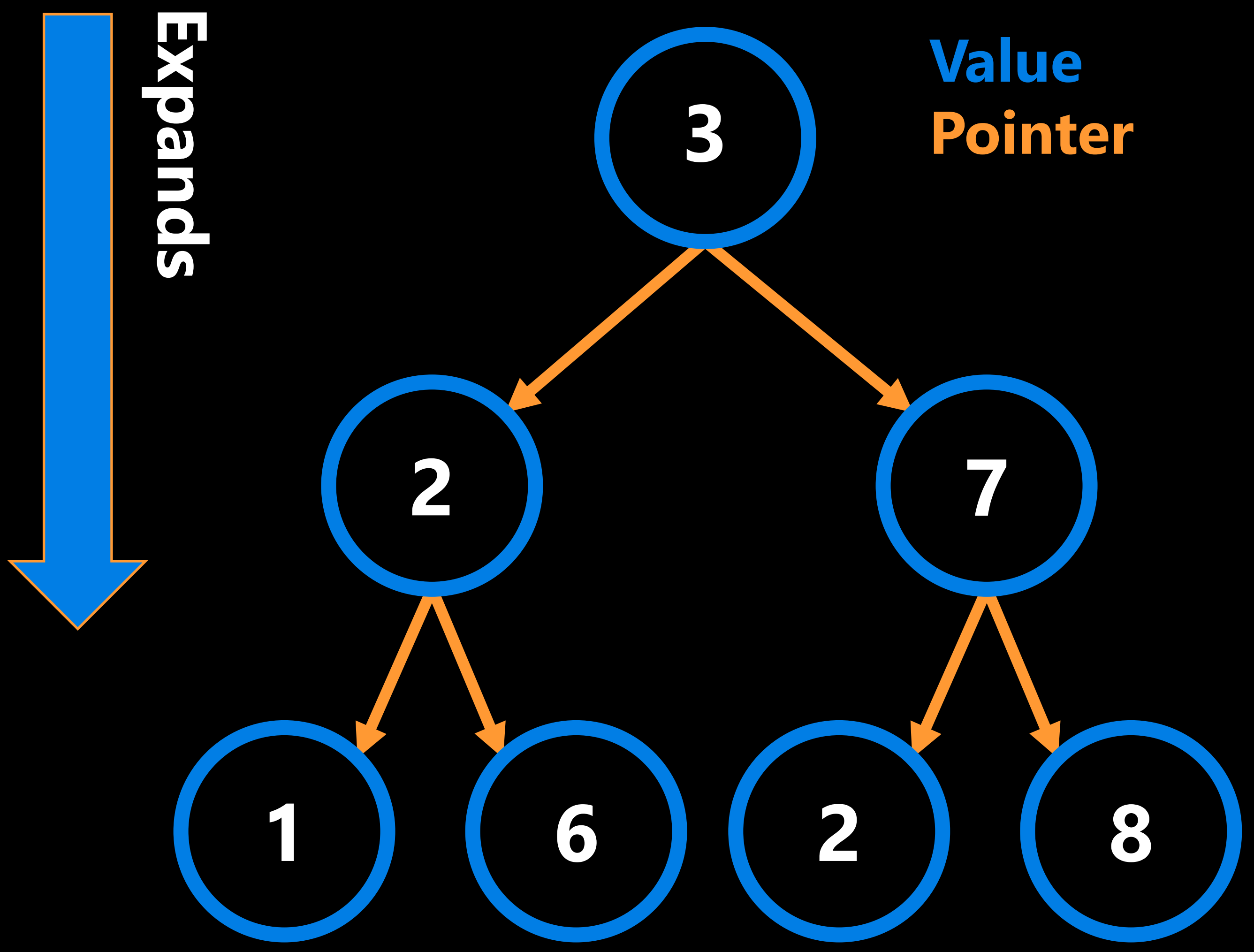


Node based Data Structures



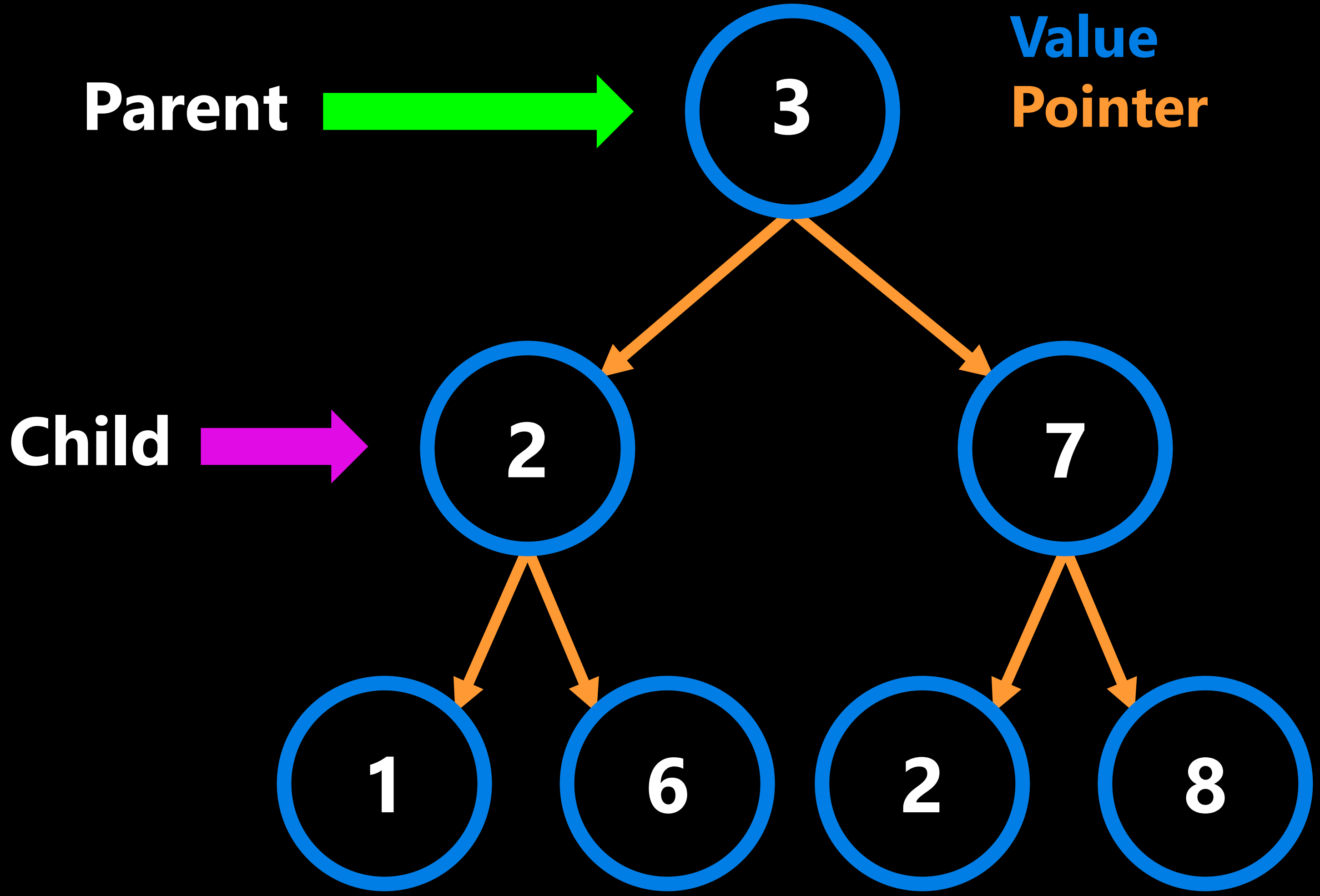
Trees

- **Trees expand in one direction.**
- Trees are made up of parents and children.
  - These are relative terms for nodes.
  - Every parent can be a child.



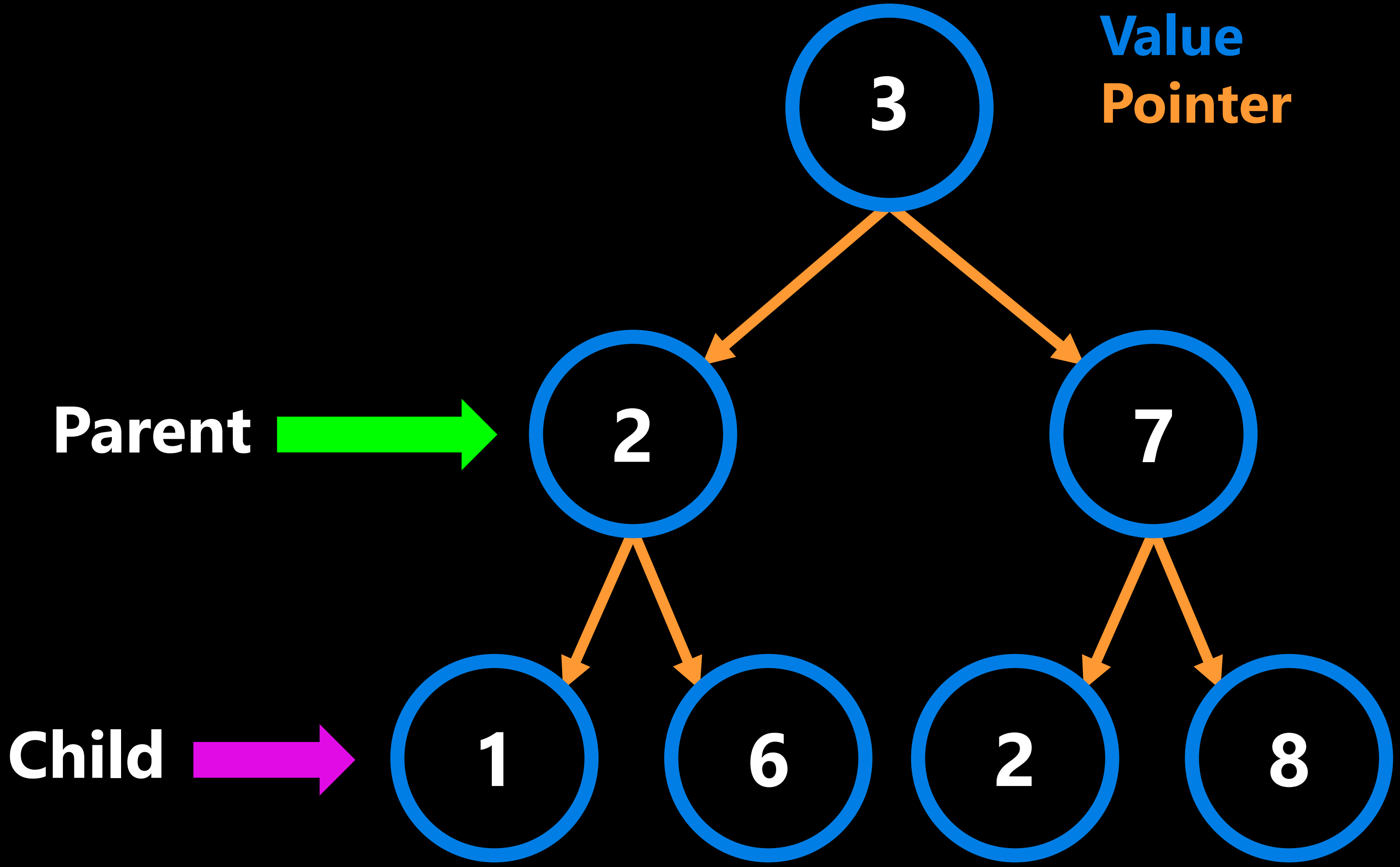
Trees

- Trees expand in one direction.
- **Trees are made up of parents and children.**
  - **These are relative terms for nodes.**
  - **Every parent can be a child.**



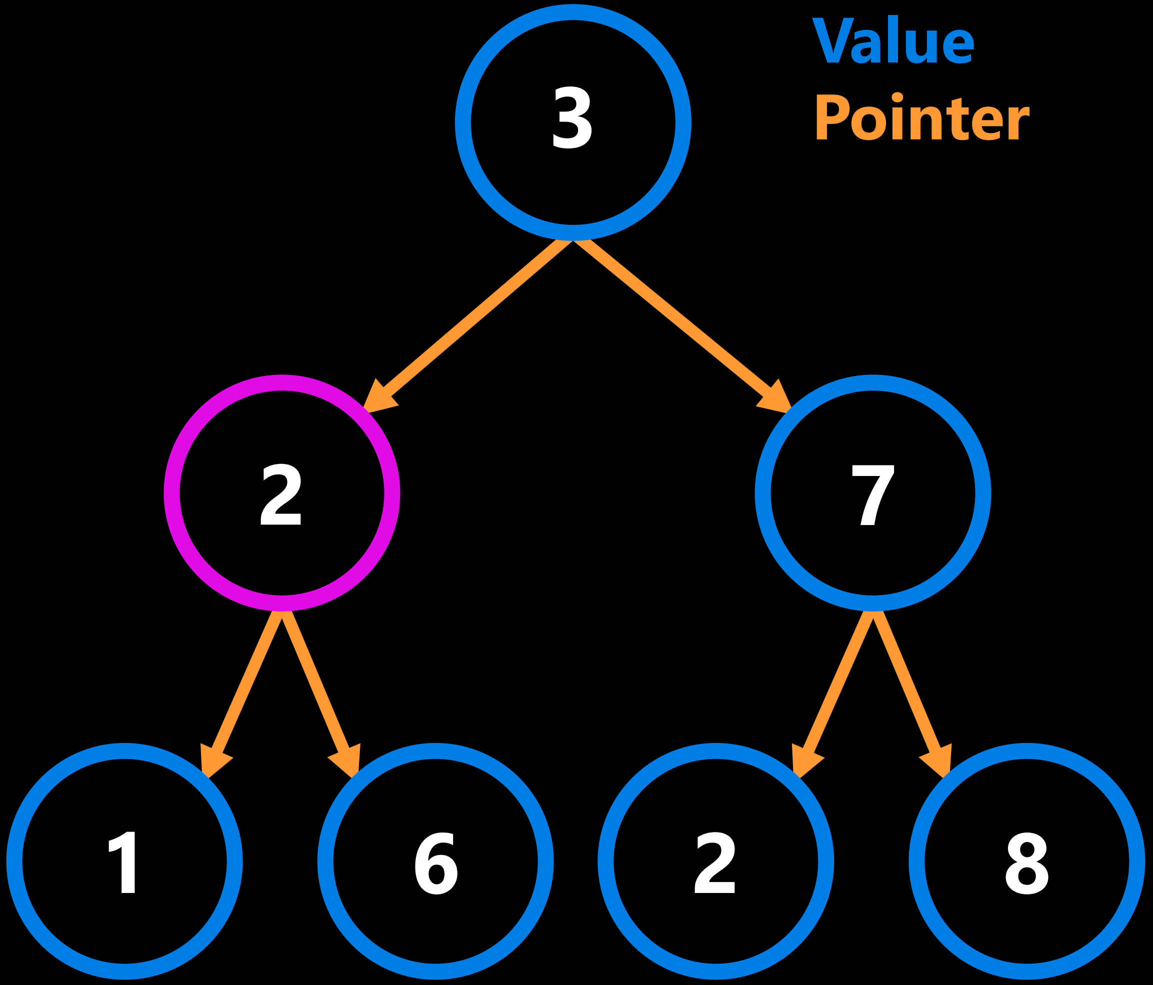
Trees

- Trees expand in one direction.
- **Trees are made up of parents and children.**
  - **These are relative terms for nodes.**
  - **Every parent can be a child.**



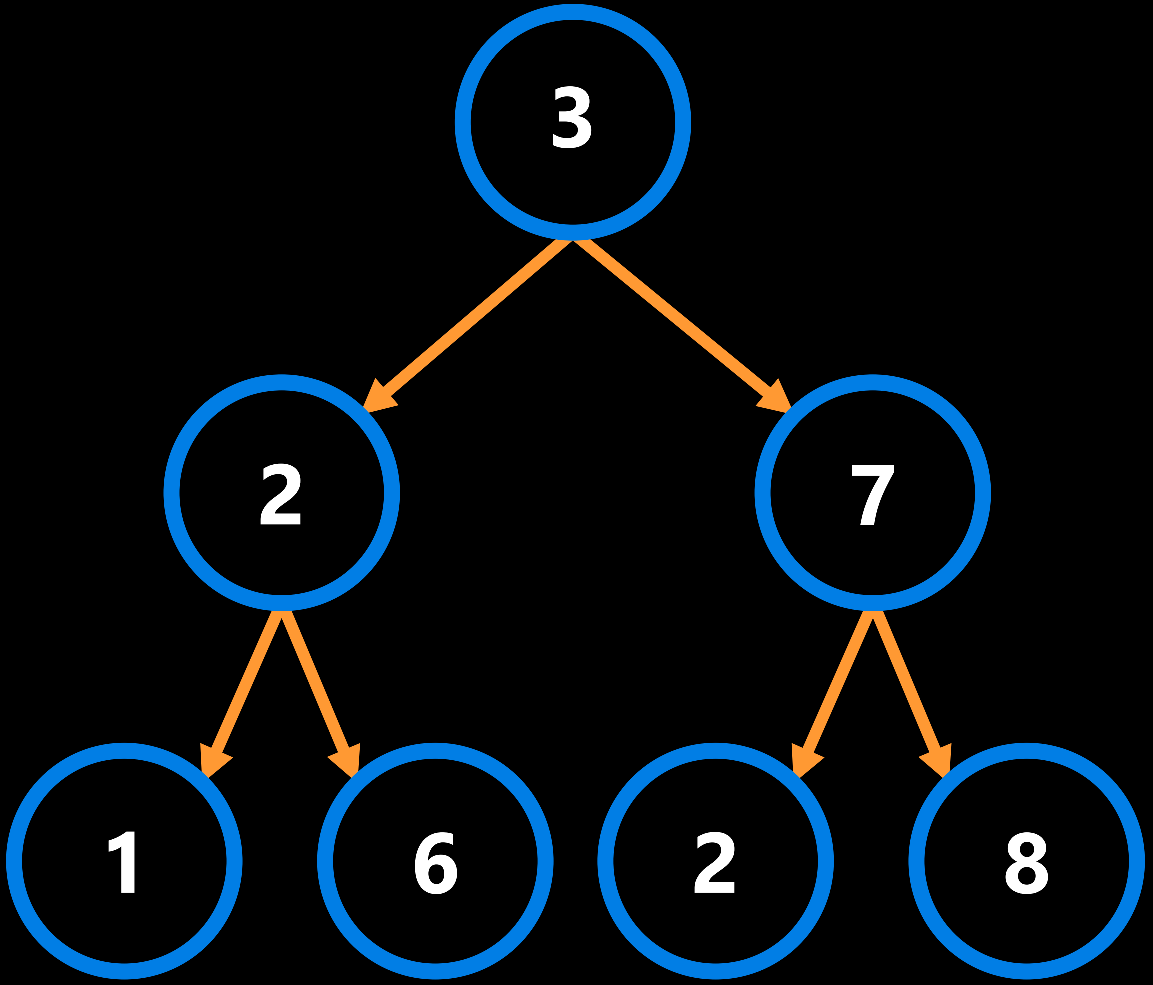
Trees

- Node 2 is a child of Node 3 and a parent of Node 1 and Node 2.
- Every node can only have one parent but can have many children.



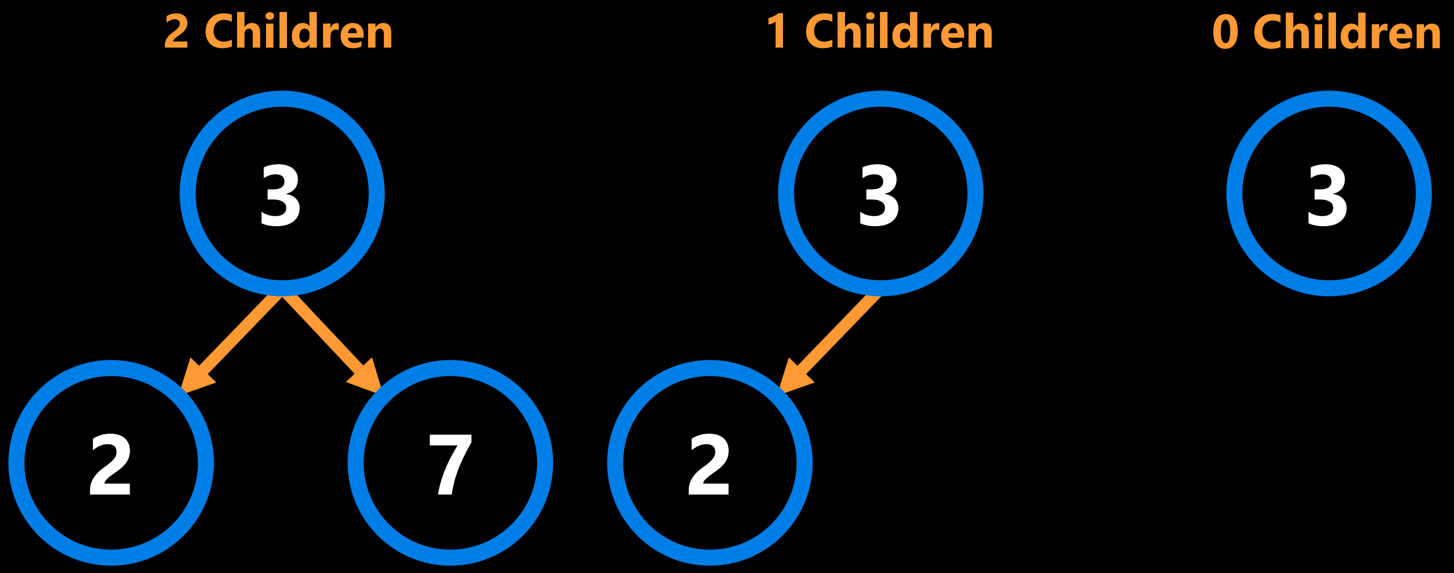
Trees

- There are many different types of trees.
  - Family Trees.
  - Decision Trees.
  - Heaps.
  - Tries.
  - HTML Trees.
  - **Binary Trees (We will focus on these).**



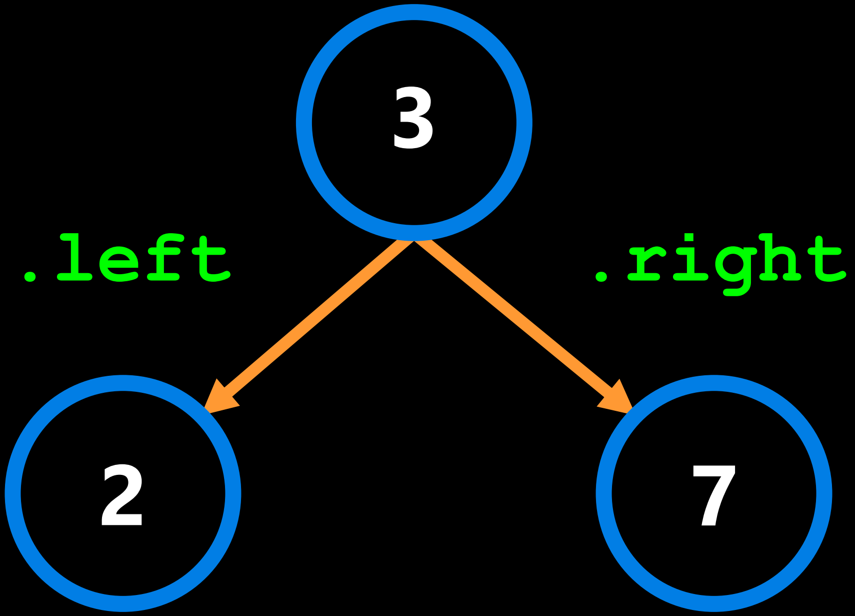
Binary Trees

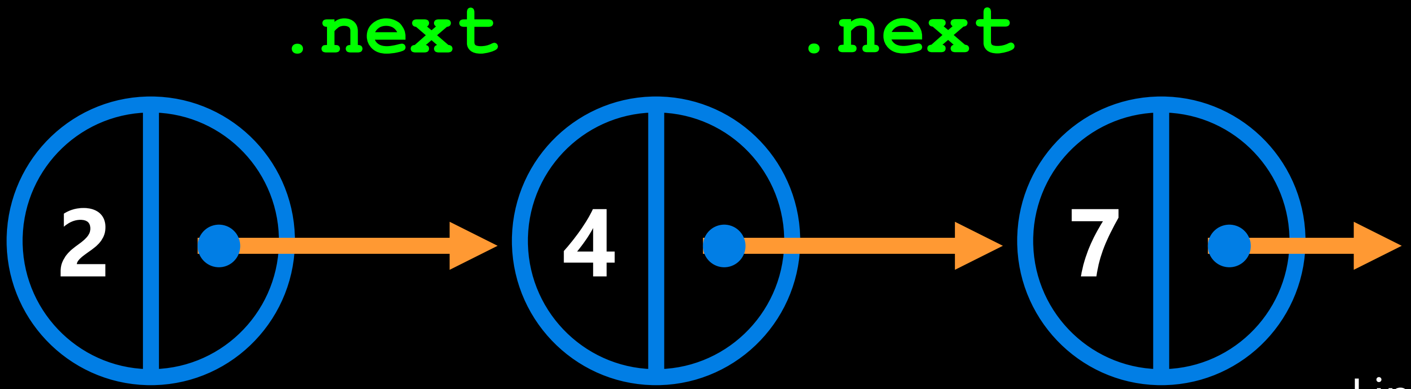
- Main Rule:
  - Each Node can have a maximum of two children (Pointers).
    - **0 Children**
    - **1 Children**
    - **2 Children**



Binary Trees

- Children are represented using `.left` and `.right`.

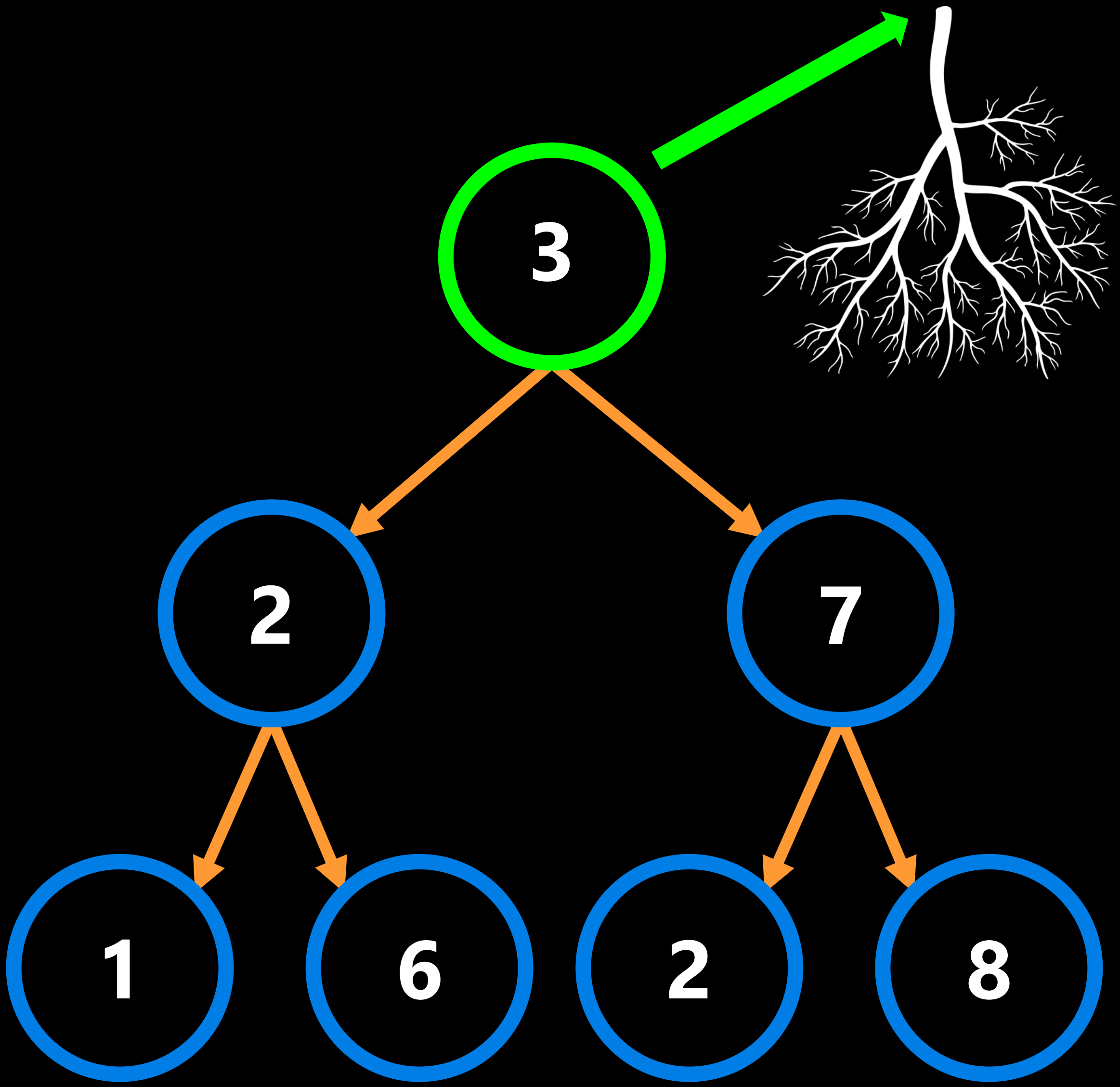




Linked list for reference.

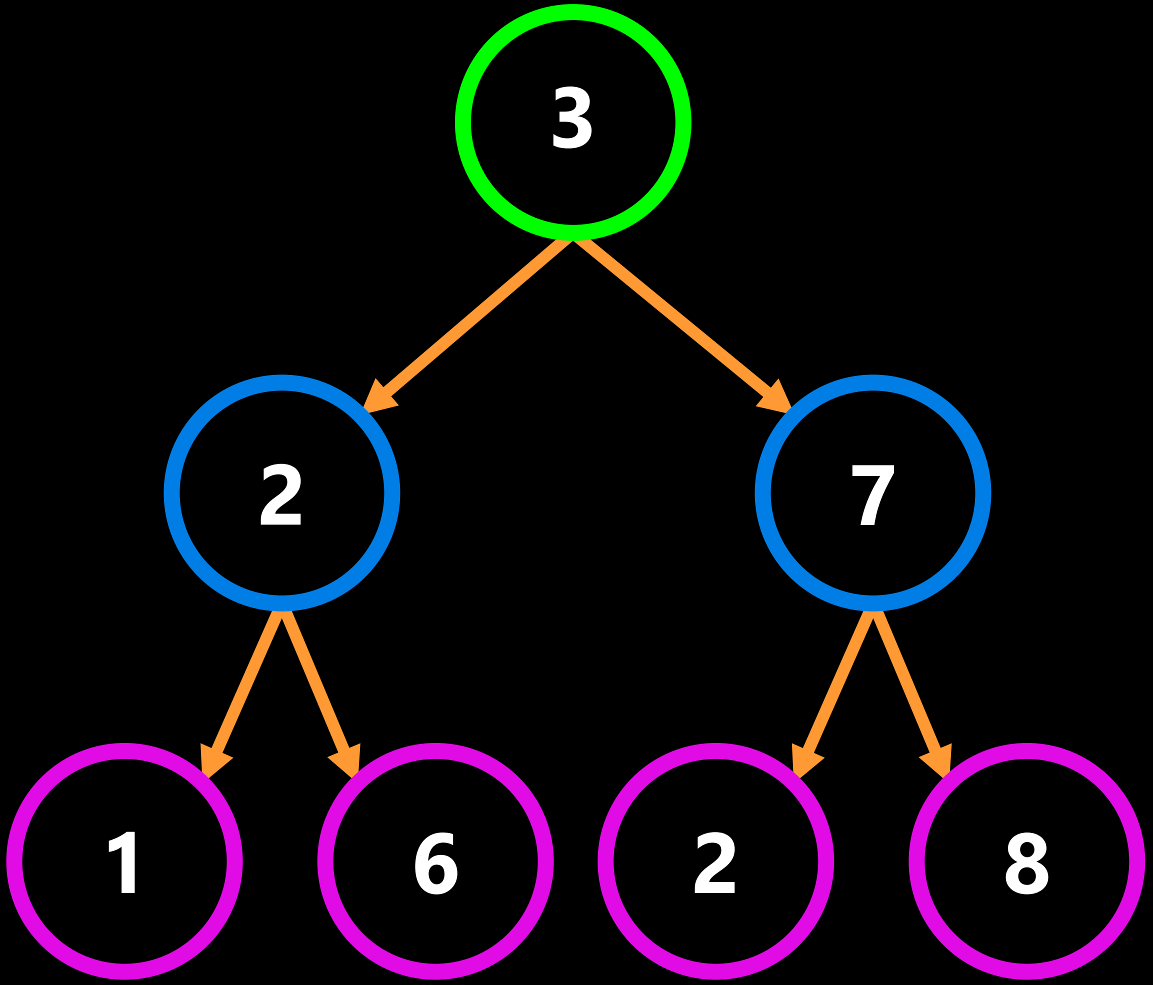
Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a leaf node.
- The path between the root node and a leaf node is called a branch.



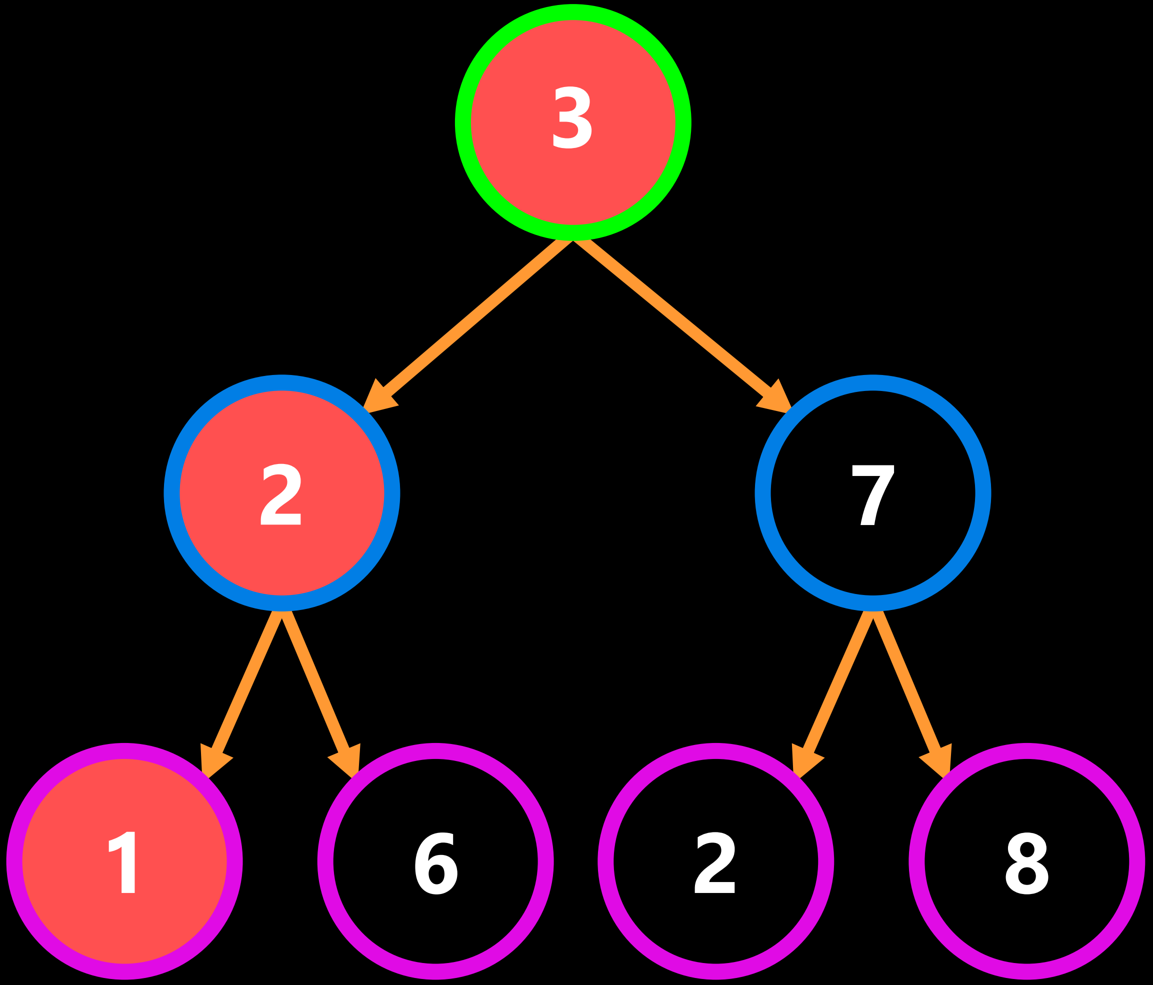
Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a **leaf node**.
- The path between the root node and a leaf node is called a branch.



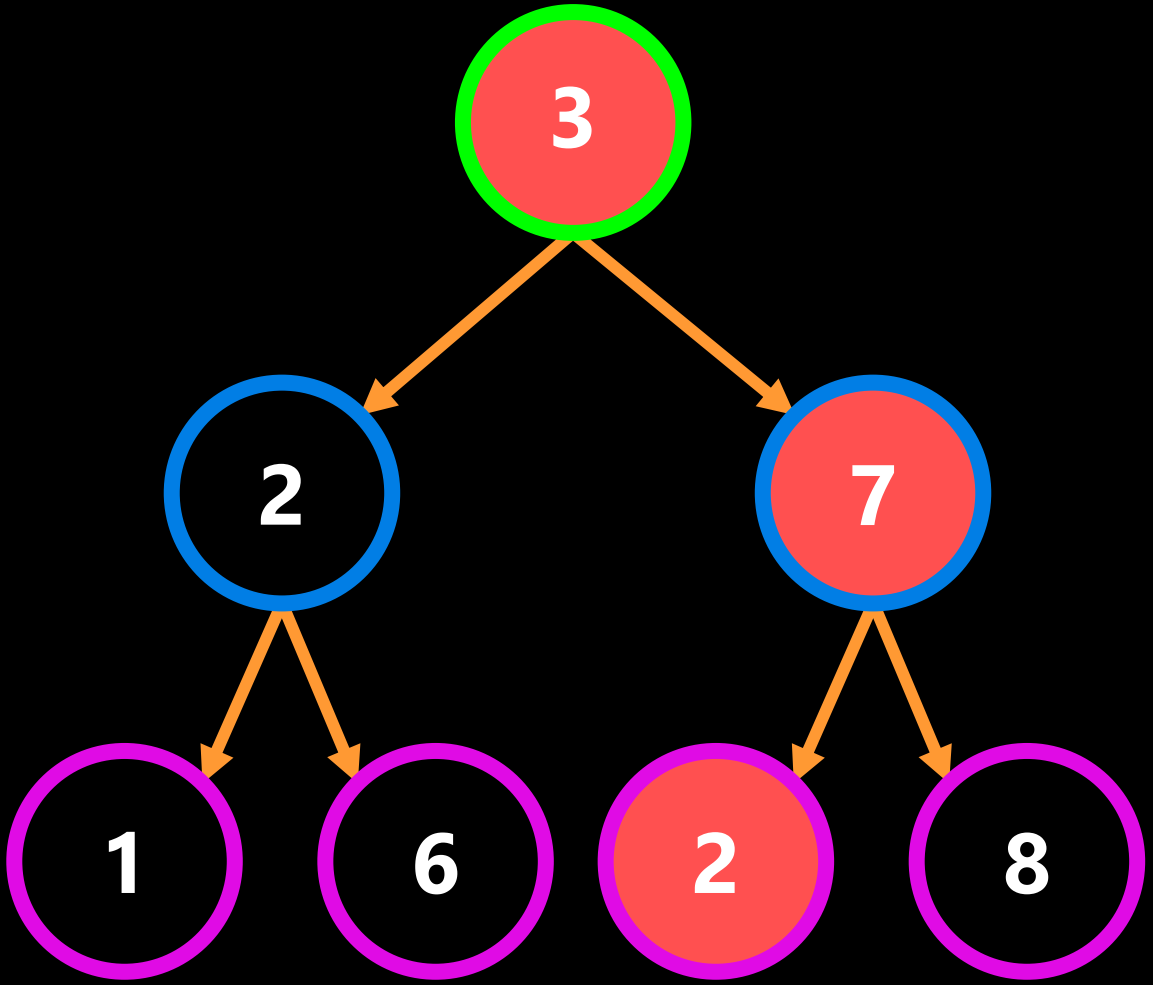
Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a **leaf node**.
- The path between the root node and a leaf node is called a **branch**.



Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a **leaf node**.
- The path between the root node and a leaf node is called a **branch**.



The Tree Node Class

- Let's check out the TreeNode class functionality.

Open your notebook

Click Link:
4. TreeNode Class

The Binary Tree Class

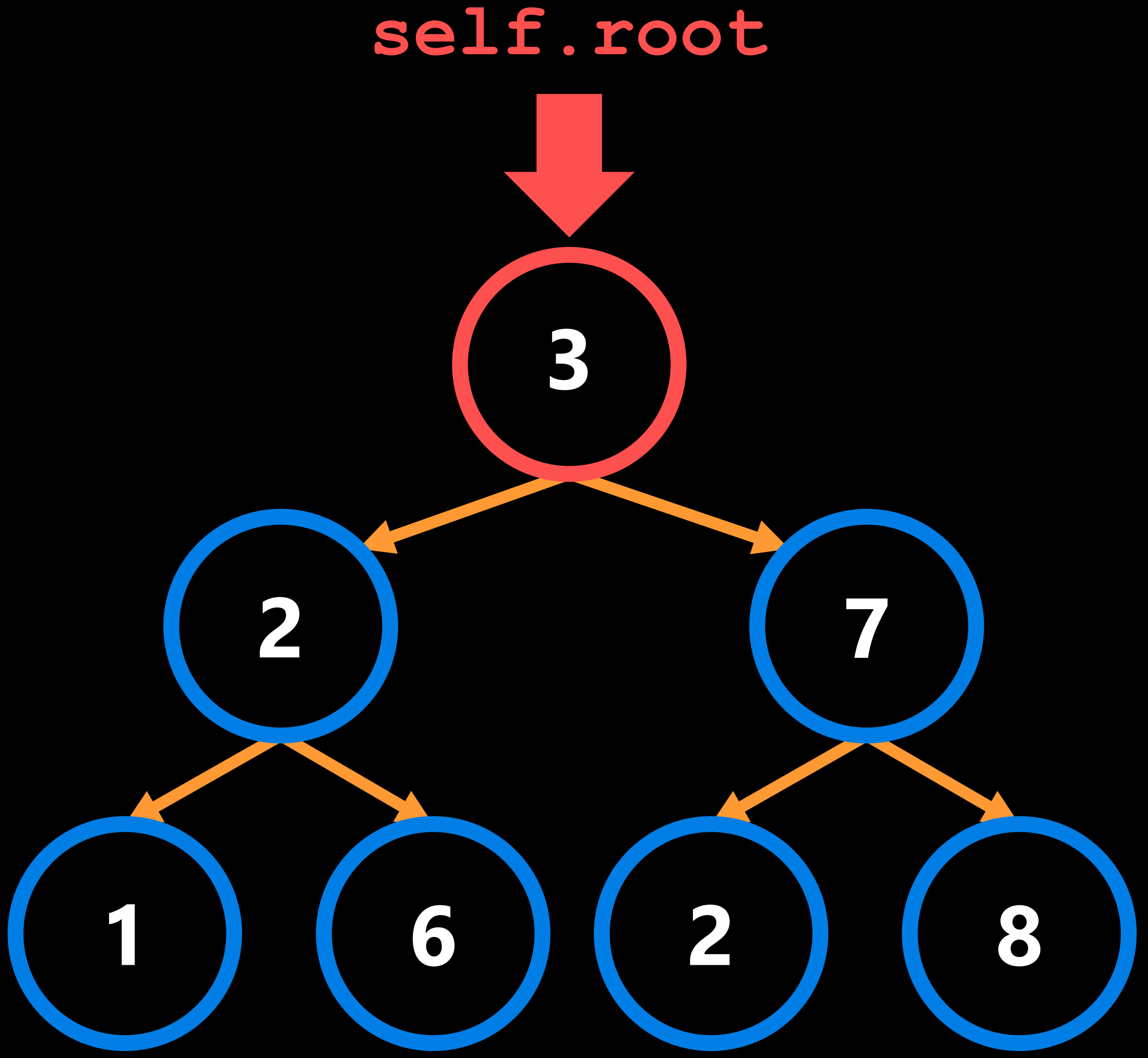
- Let's check out the BinaryTree class functionality.

Open your notebook

Click Link:
5. BinaryTree Class

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

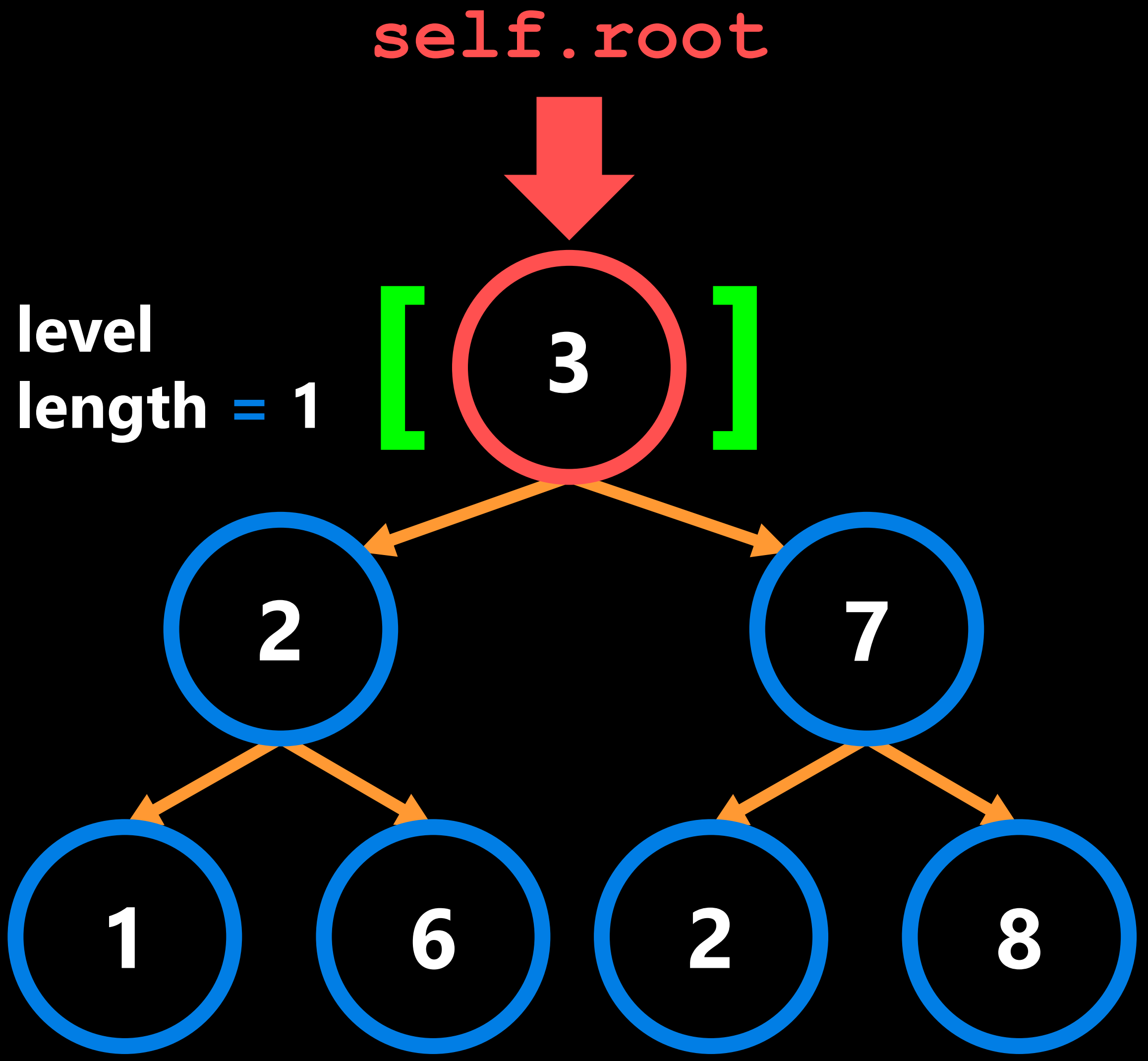


```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

Create level list.

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
```

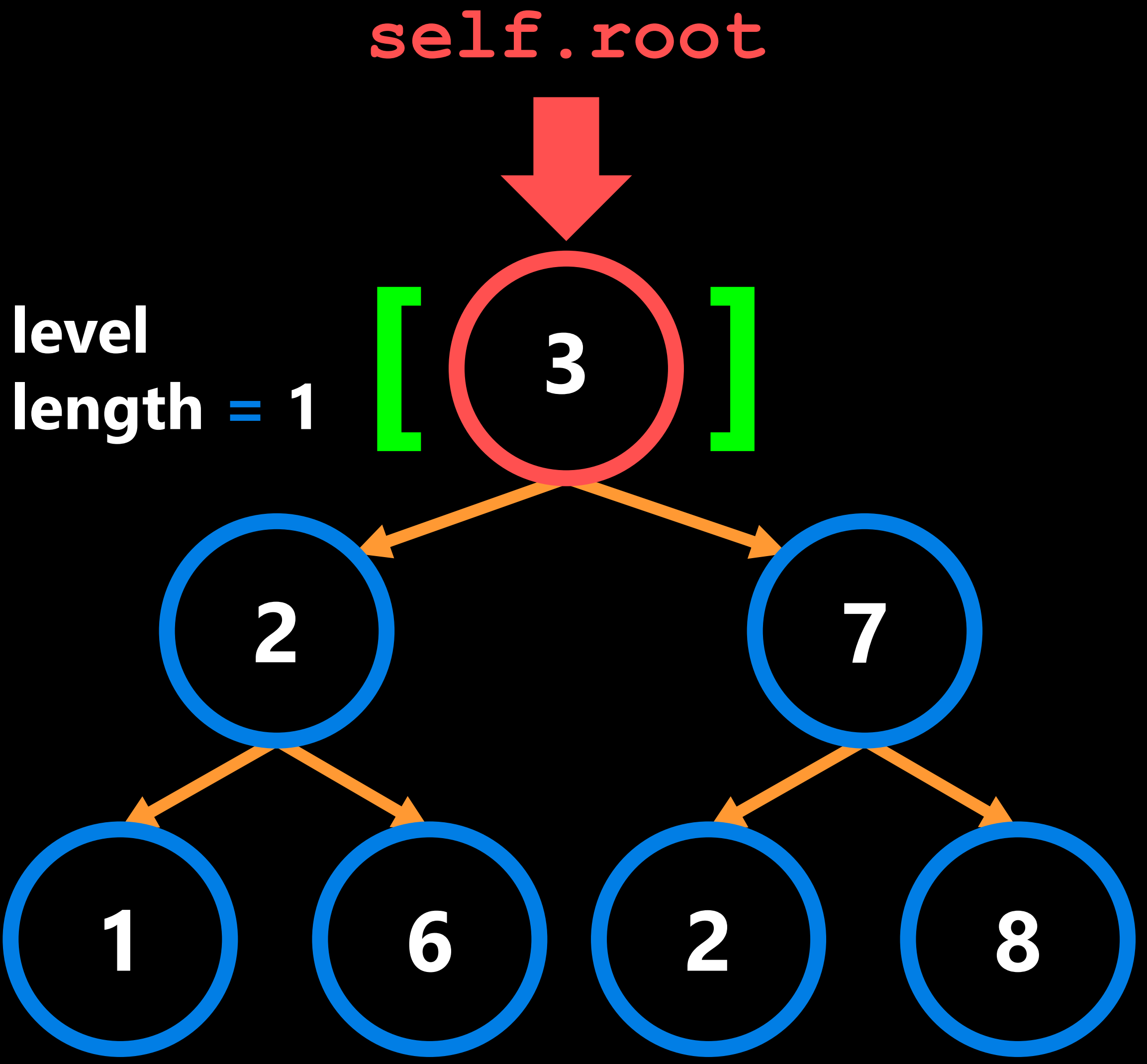


level
length = 1

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                          TreeNode(2, TreeNode(1), TreeNode(6)),
                          TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
```



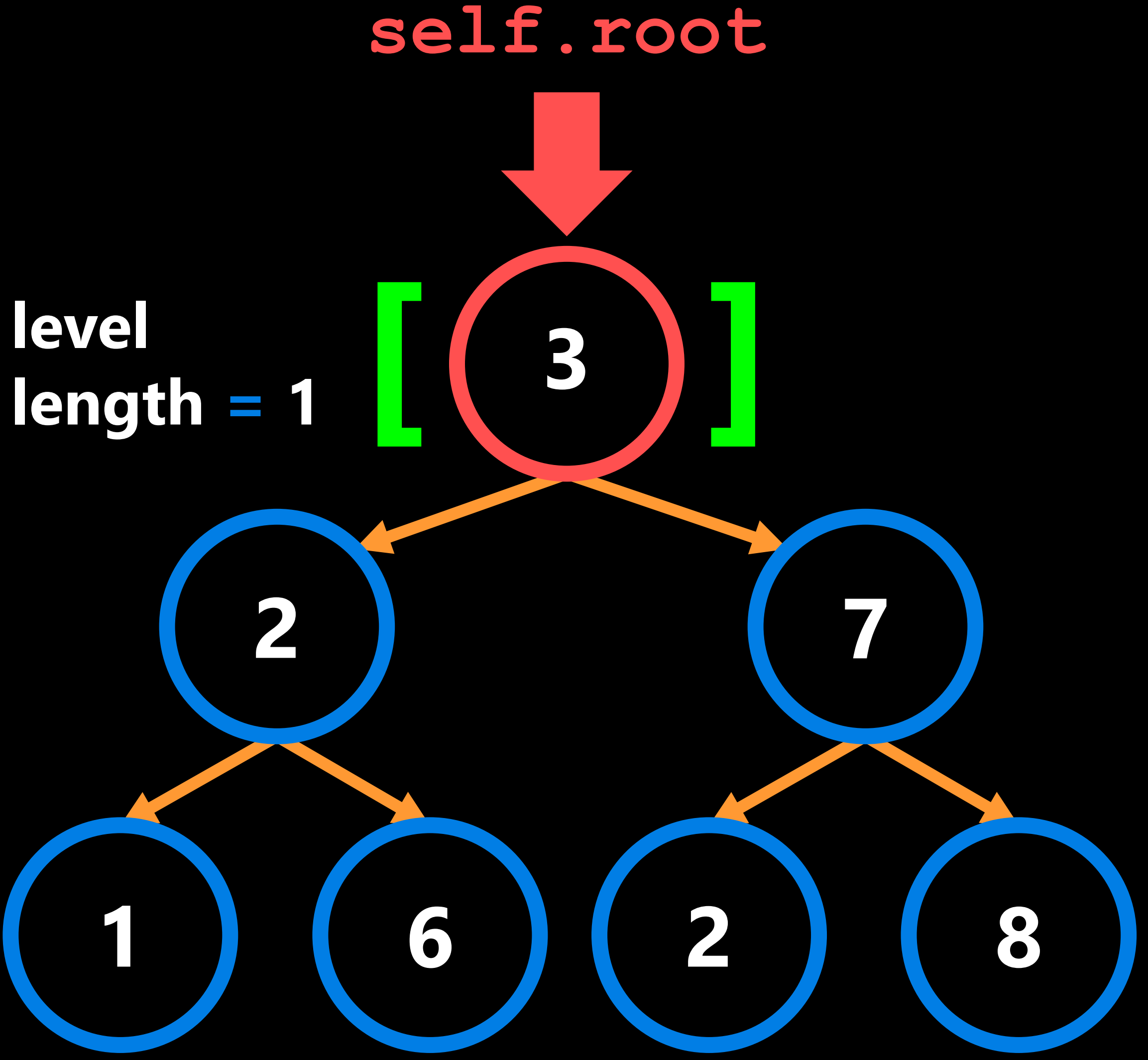
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

Create empty list.

level_next = [ ]

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
```



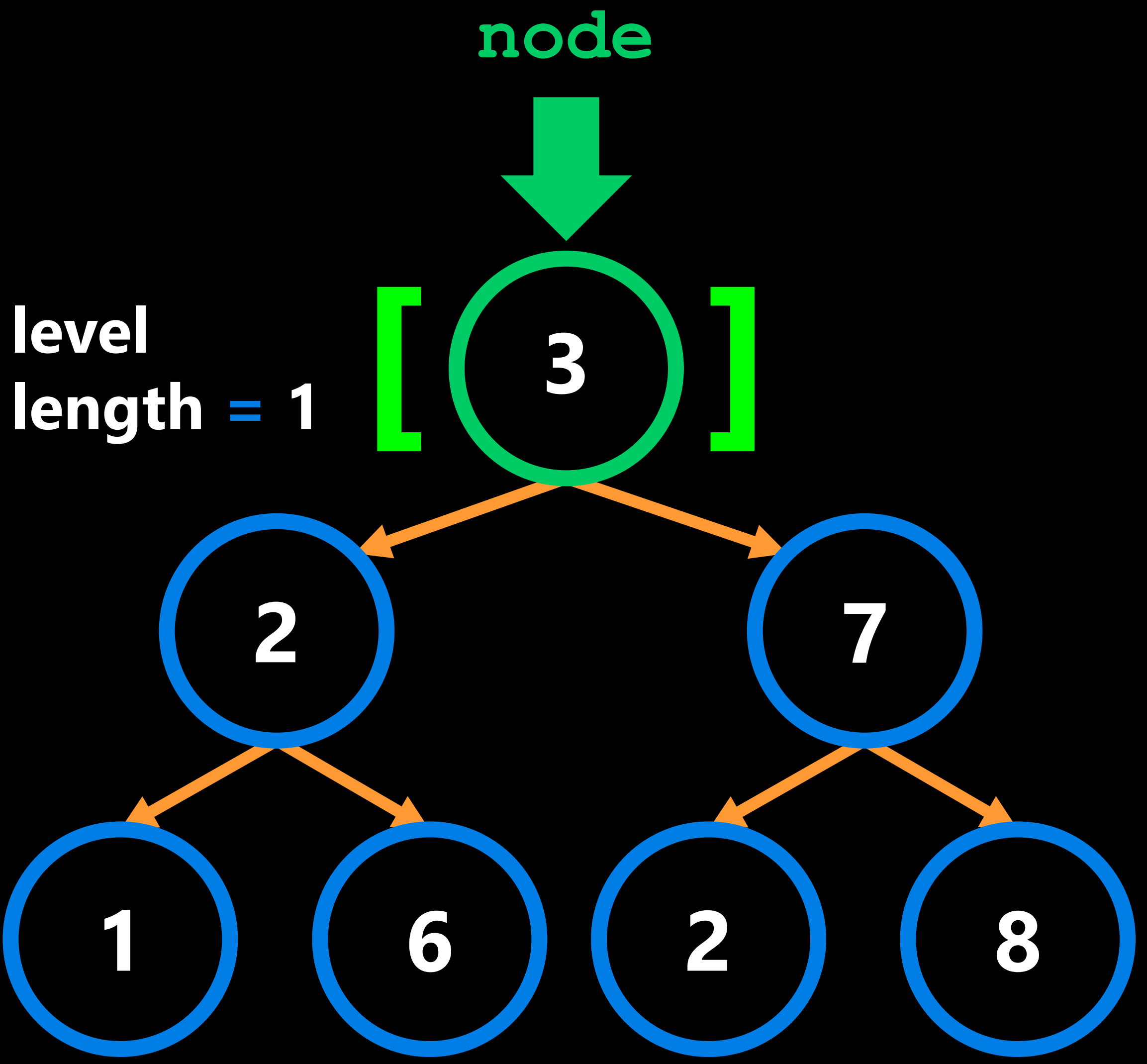
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
```

level_next = [ ]

Loop through nodes in level.



level length = 1

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

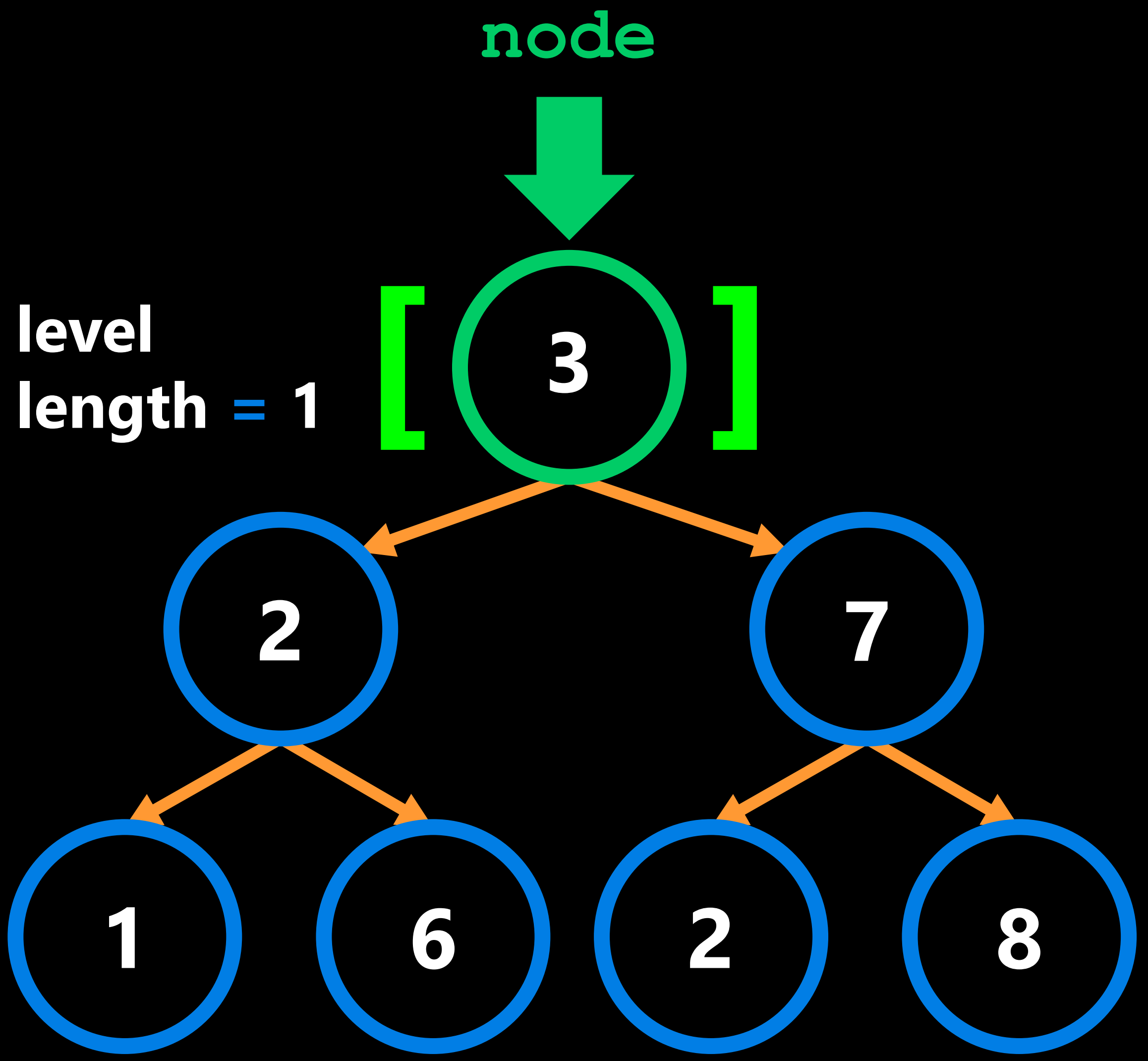
True ▶ while len(level) > 0:

print.

level_next = [ ]

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

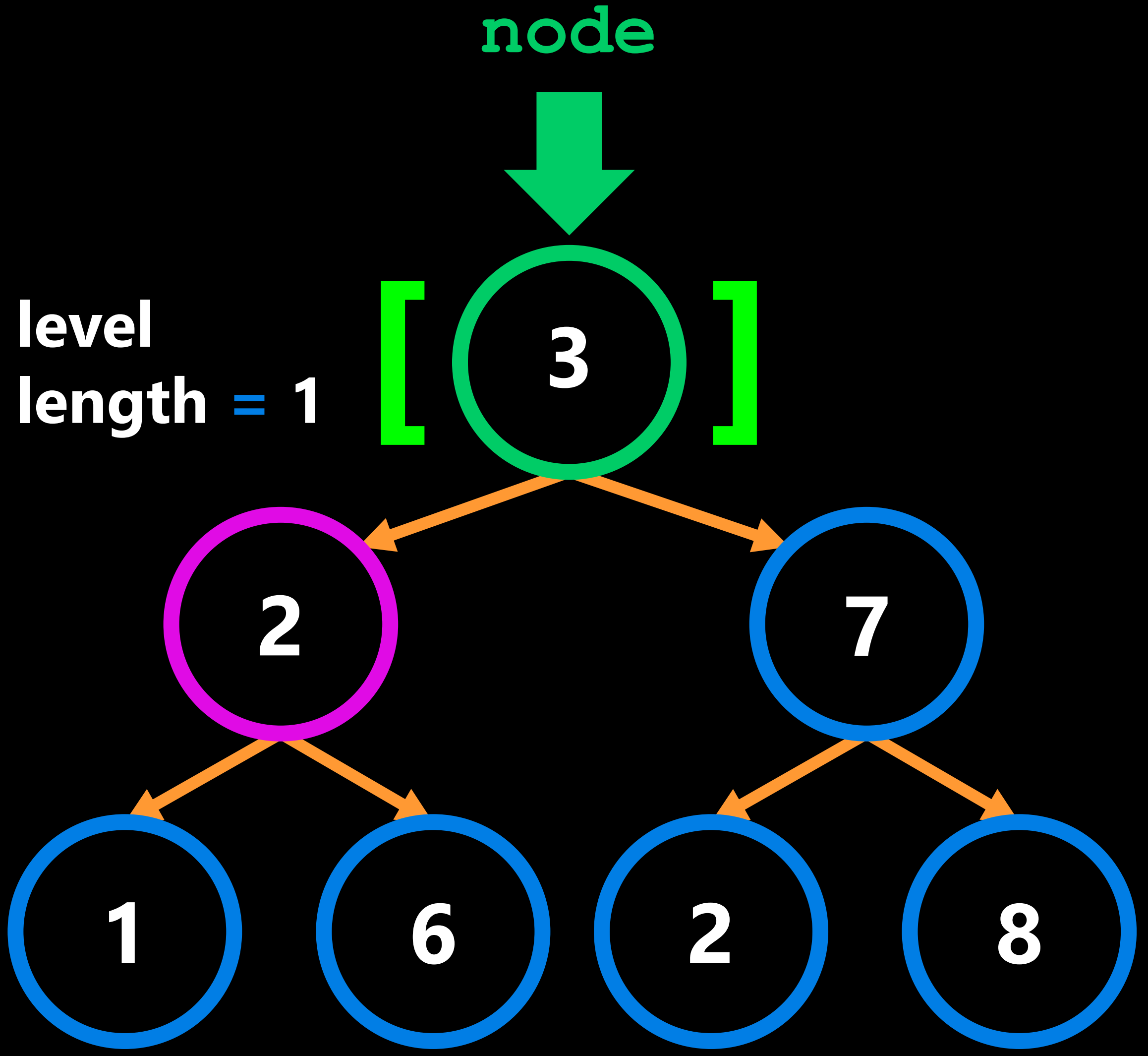


```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:  ◀ True
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

level_next = [ ]



level
length = 1

```python
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

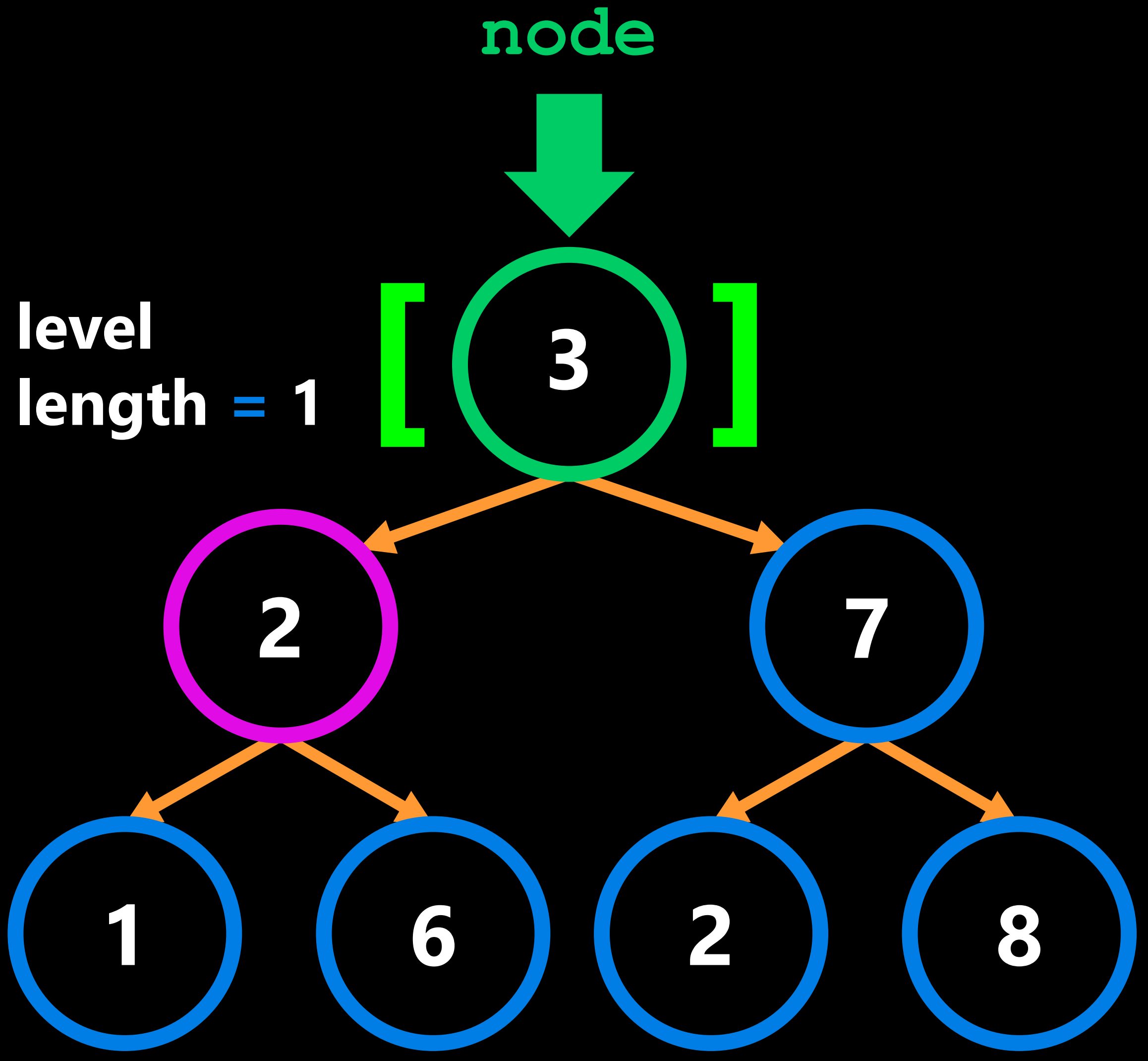
if node.left is not None: ◀ True

level_next.append(node.left) ◀ Get node.

```python
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

level_next = [2]



level
length = 1

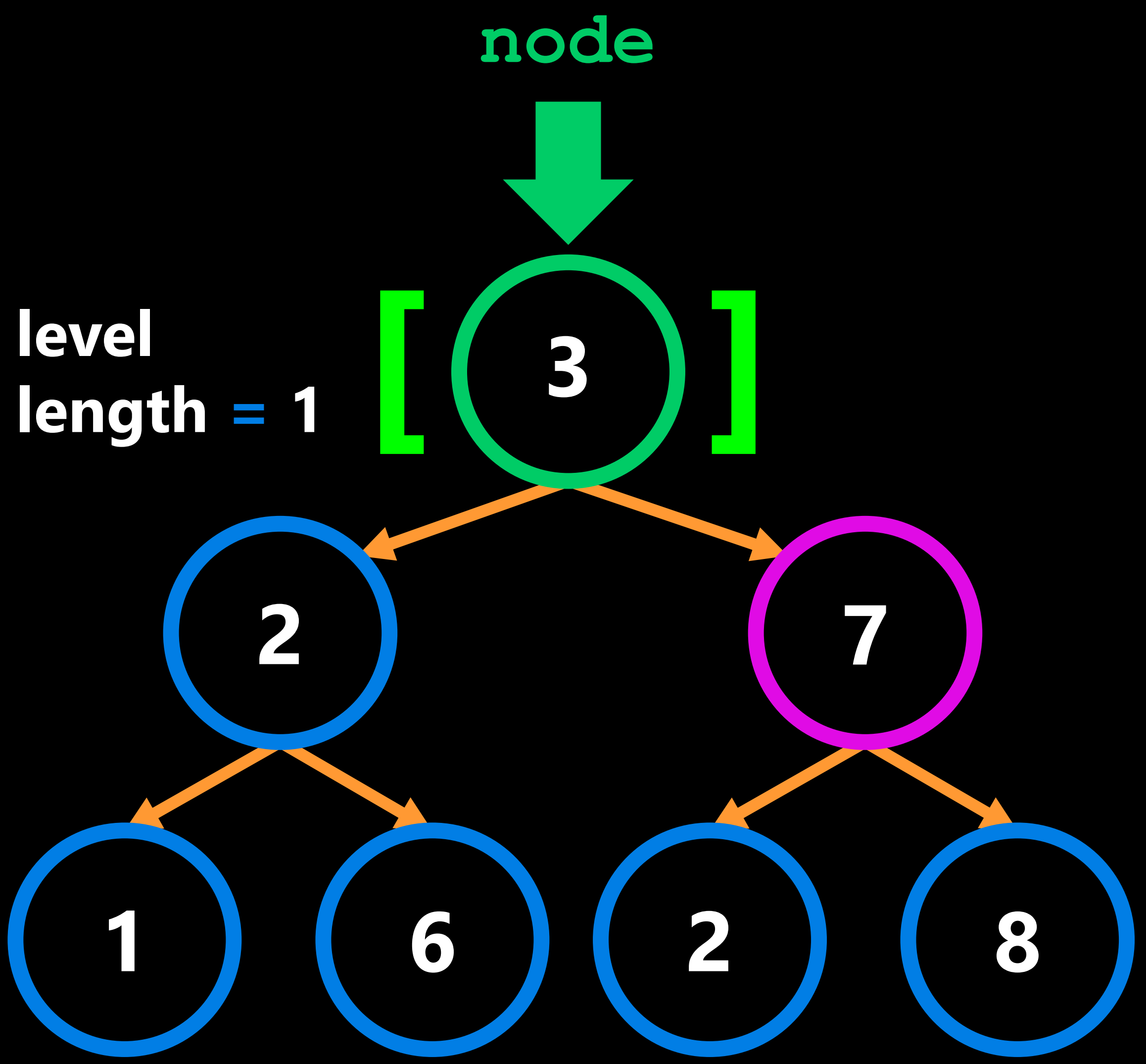
[3]

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None: ◀ True
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

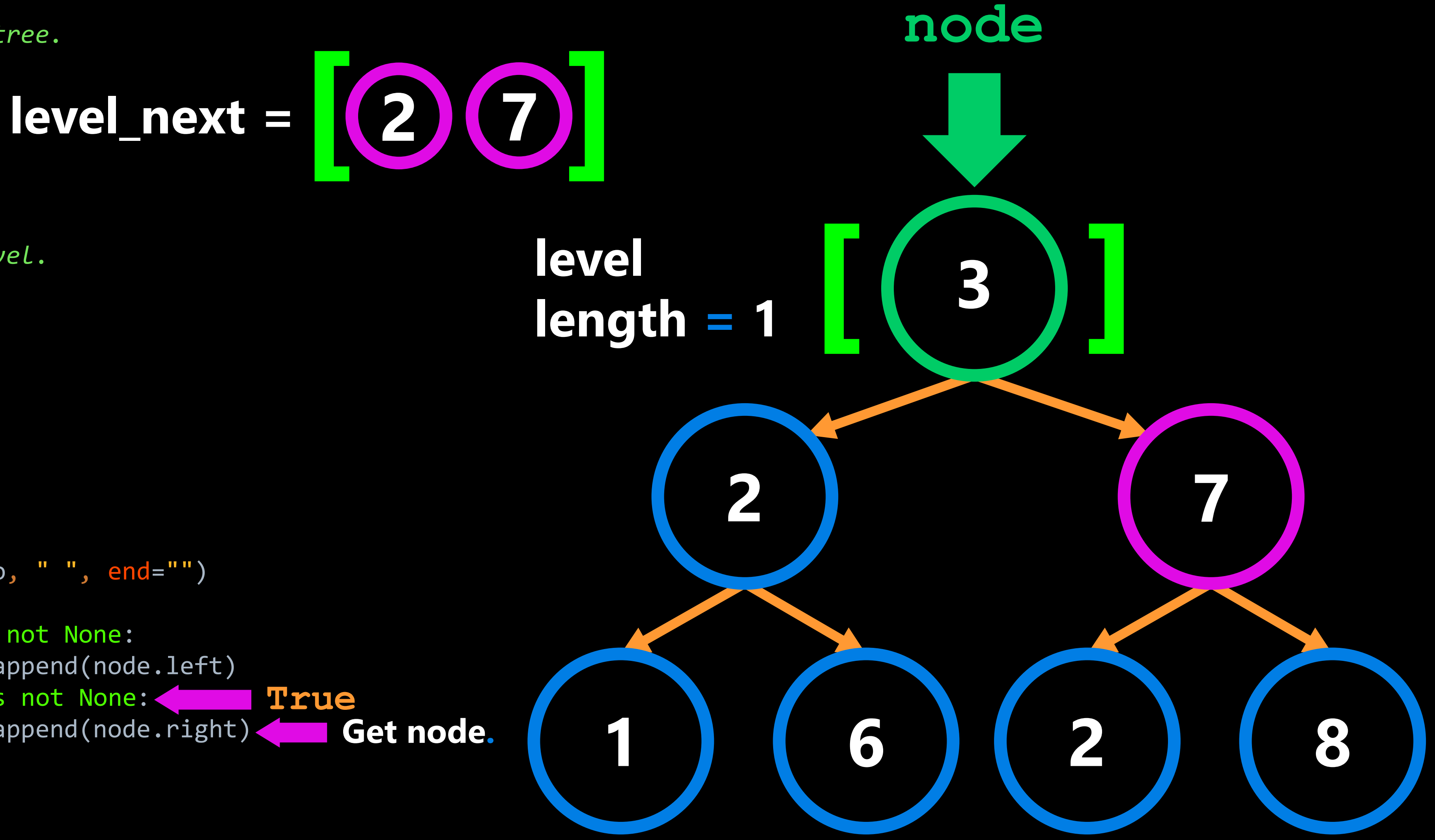
level_next = [2]



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:   ◀ True
                    level_next.append(node.right)   ◀ Get node.

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```



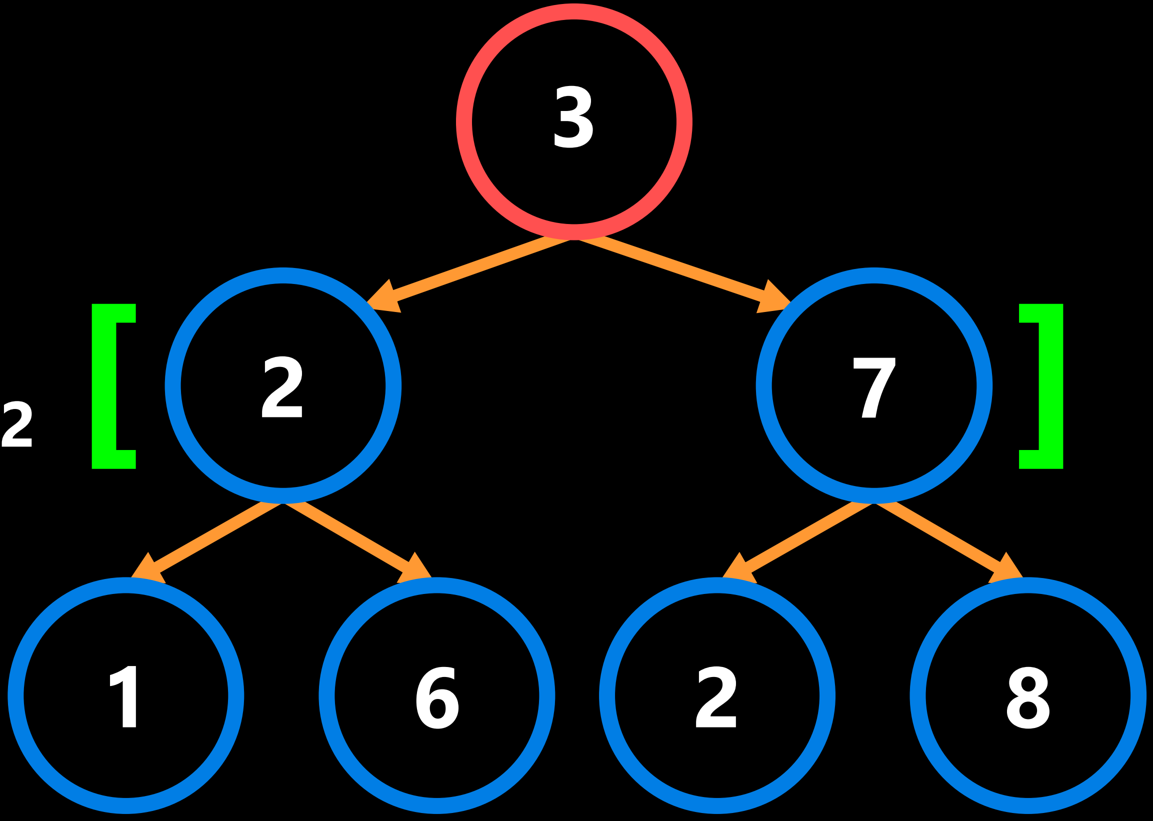
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

Move to next level.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

level_next = [2 7]



level length = 2

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

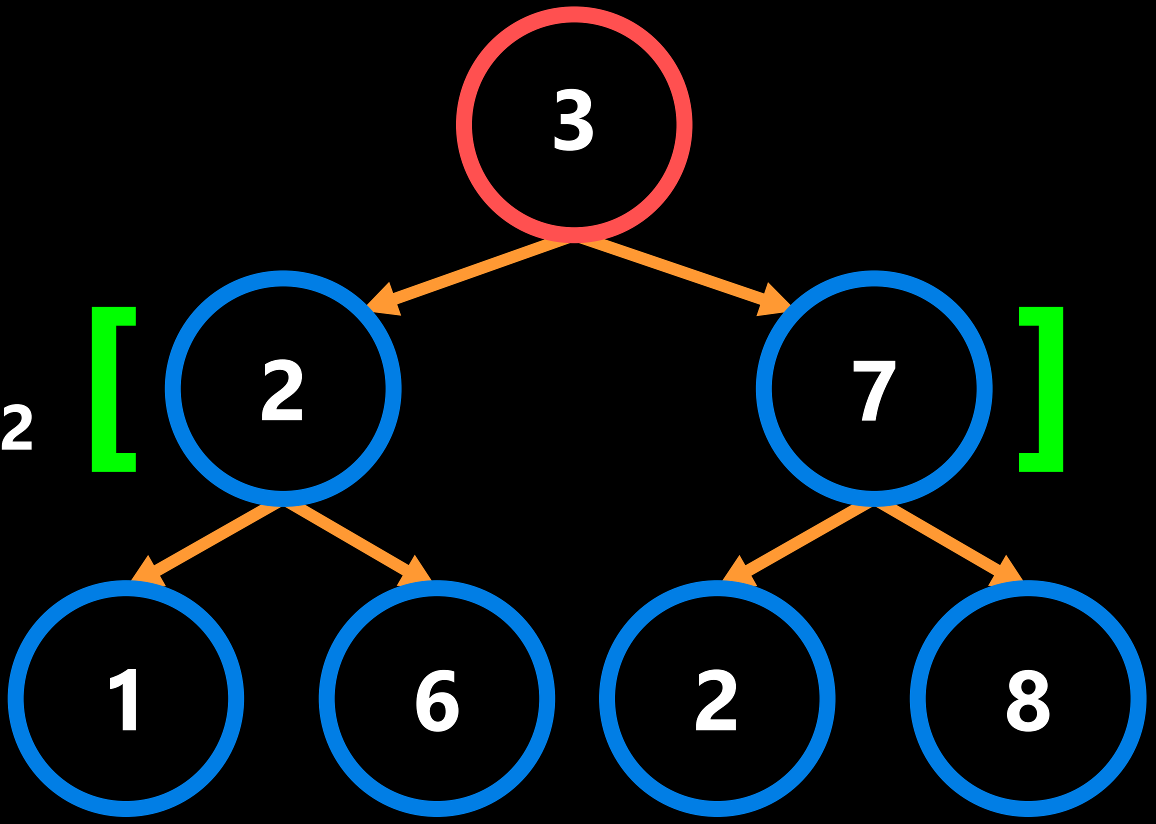
Create empty list.

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

level_next = [ ]

level length = 2



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:  ◀

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

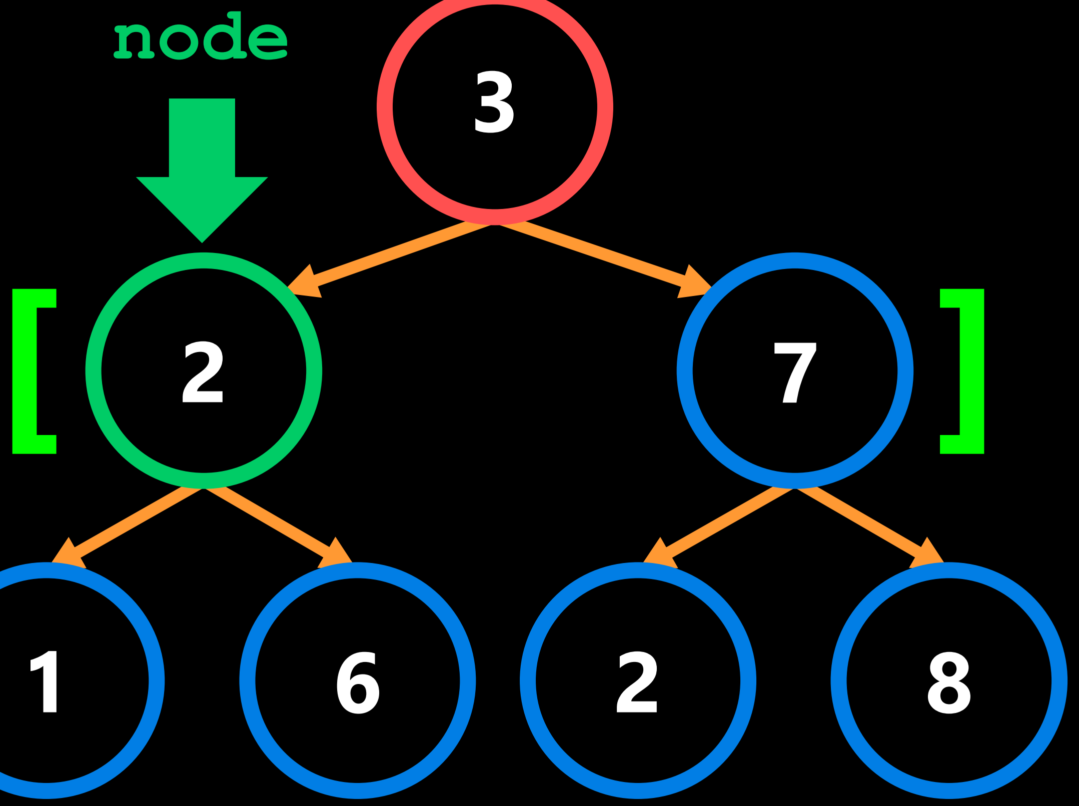
Loop through nodes in level.

```
tree = BinaryTree(TreeNode(3,
                          TreeNode(2, TreeNode(1), TreeNode(6)),
                          TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3
```

level_next = [ ]

level length = 2



```python
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

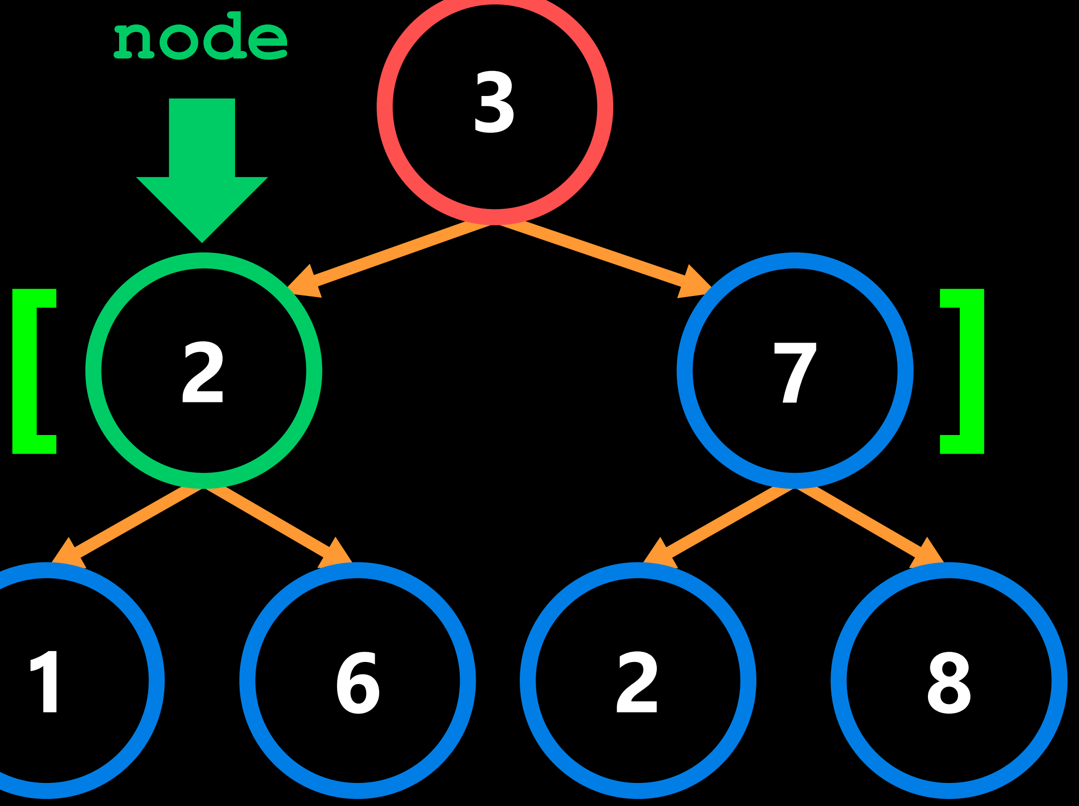
```python
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2
```

level_next = [ ]

print(node.cargo, " ", end="") ◀ **print.**

level length = 2



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

if node.left is not None: ◀ True

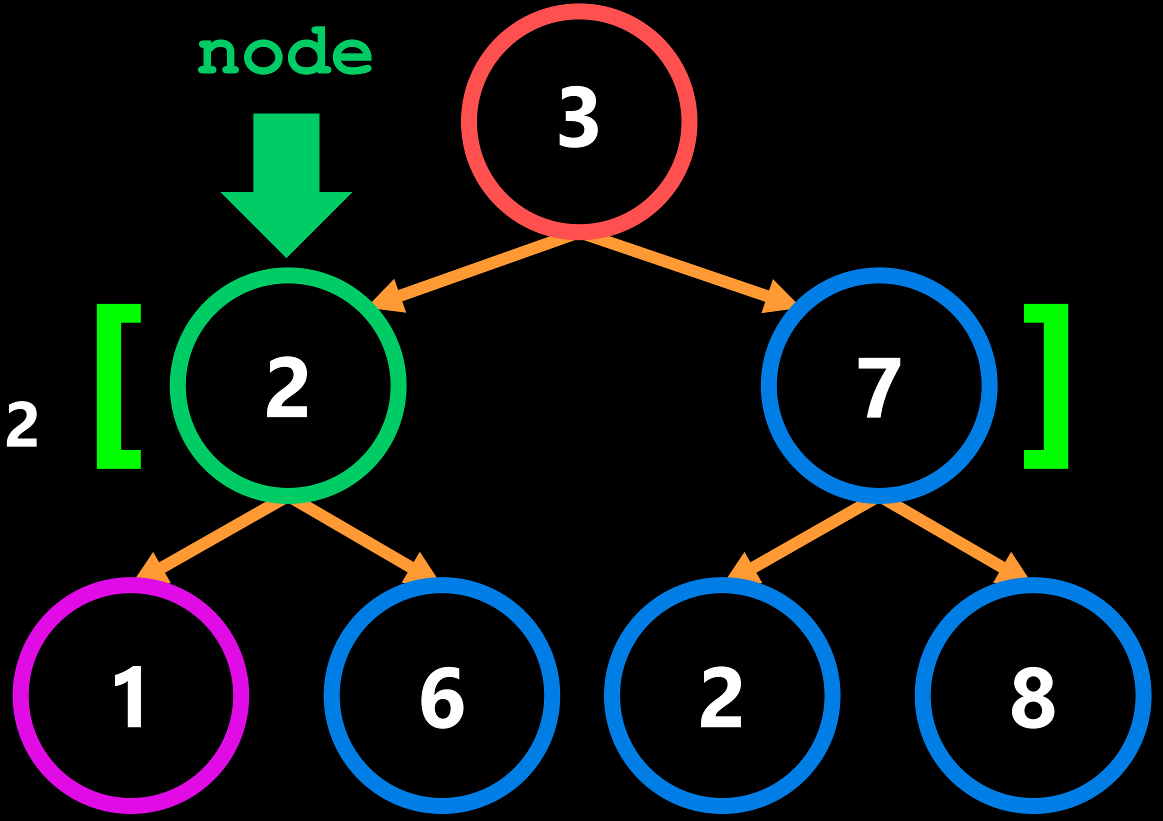
level_next.append(node.left) ◀ Get node.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2
```

level_next = [1]

level length = 2



```python
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

if node.right is not None: ◀ True

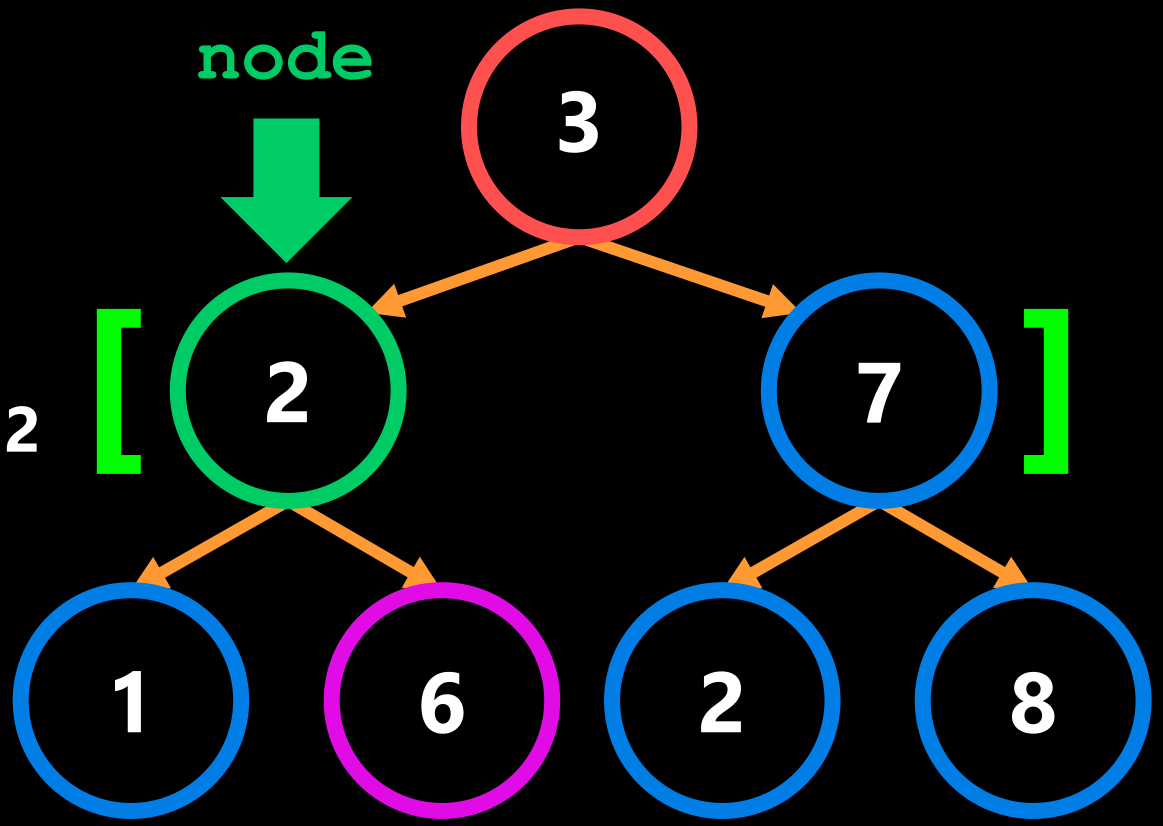
level_next.append(node.right) ◀ Get node.

```python
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2
```

level_next = [1 6]

level length = 2



```python
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

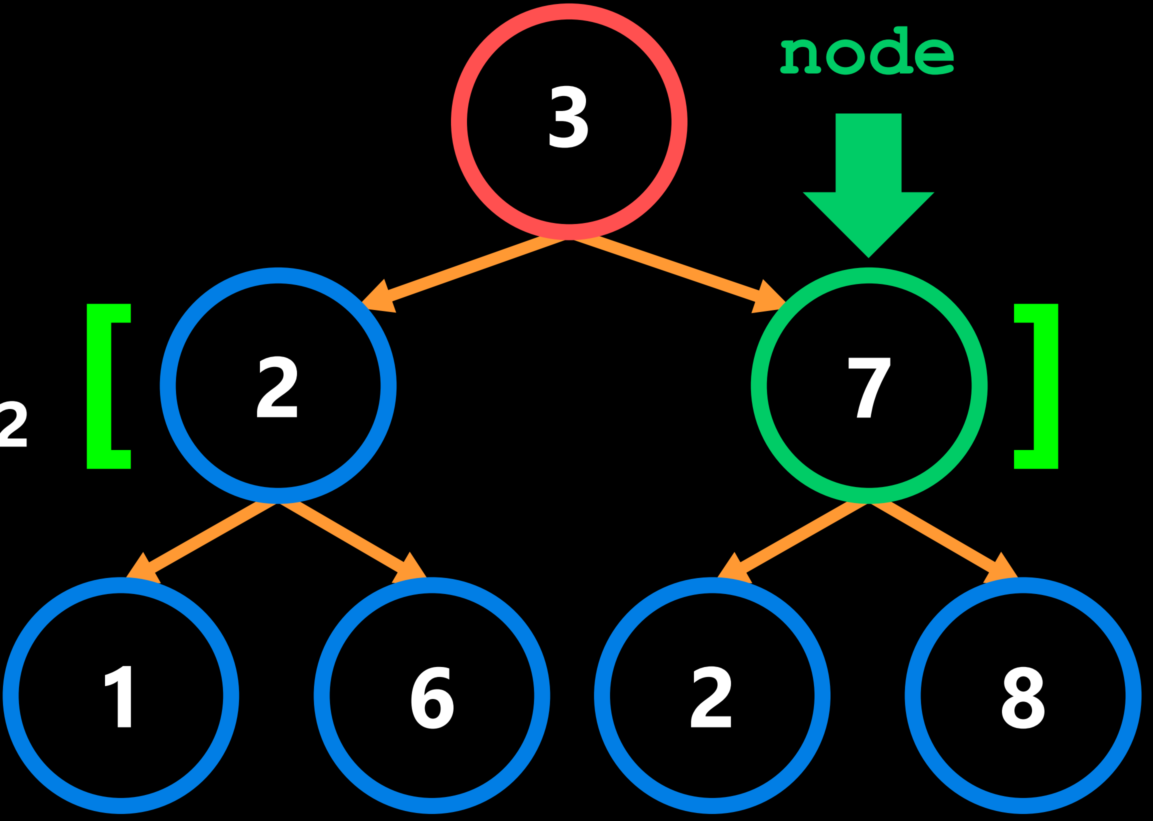
True ▶ while len(level) > 0:

Loop through nodes in level.

```python
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2
```

level_next = [1 6]



level length = 2

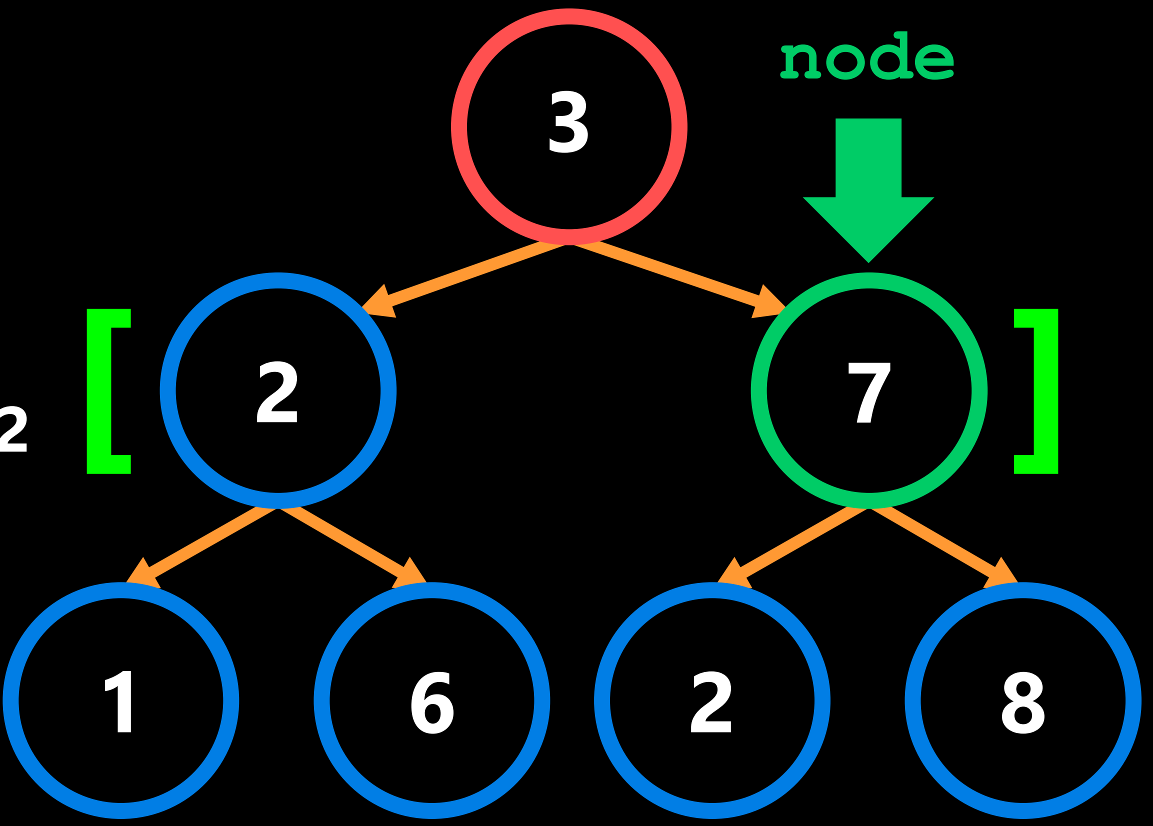
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")  ◀ print.

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7
```

level_next = [1 6]

level length = 2



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

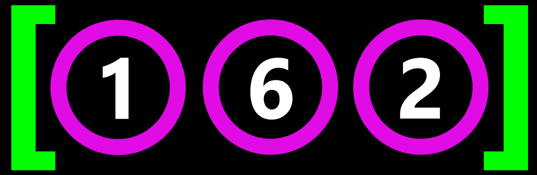
if node.left is not None: ◀ True

level_next.append(node.left) ◀ Get node.

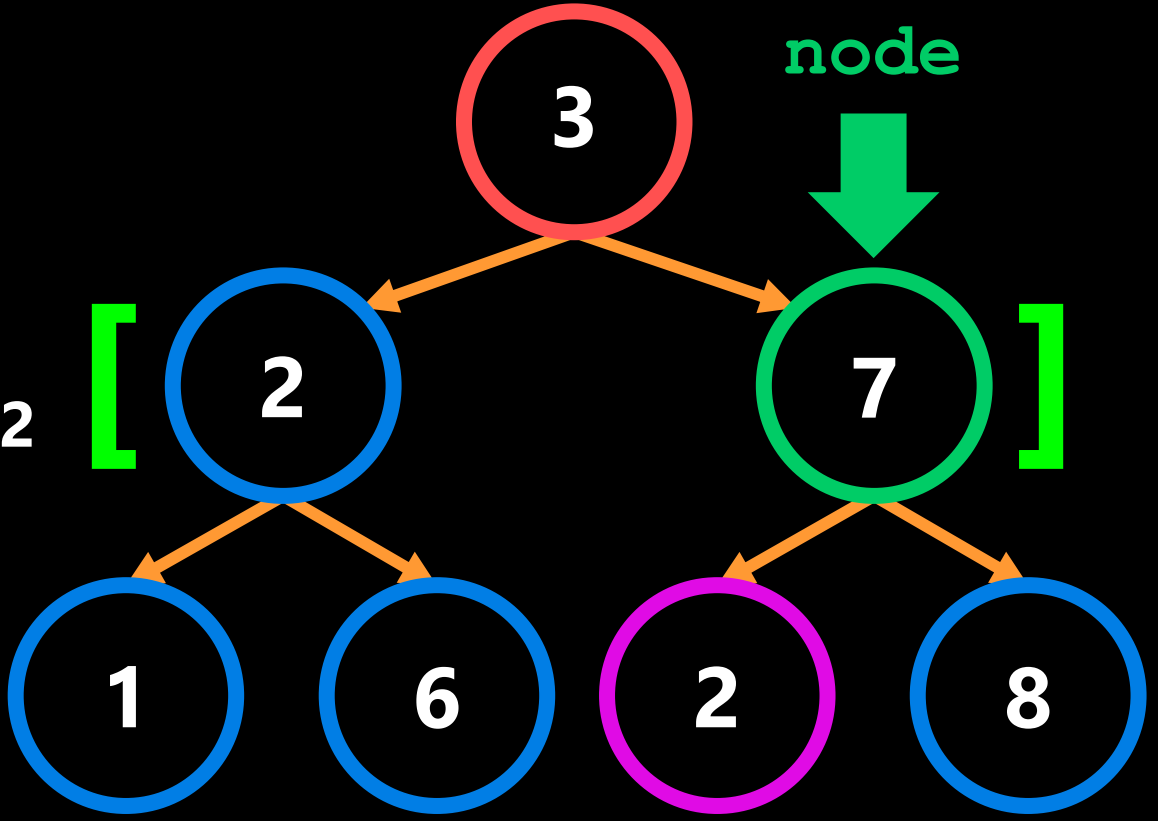
```
tree = BinaryTree(TreeNode(3,
                          TreeNode(2, TreeNode(1), TreeNode(6)),
                          TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7
```

level_next =



level
length = 2



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

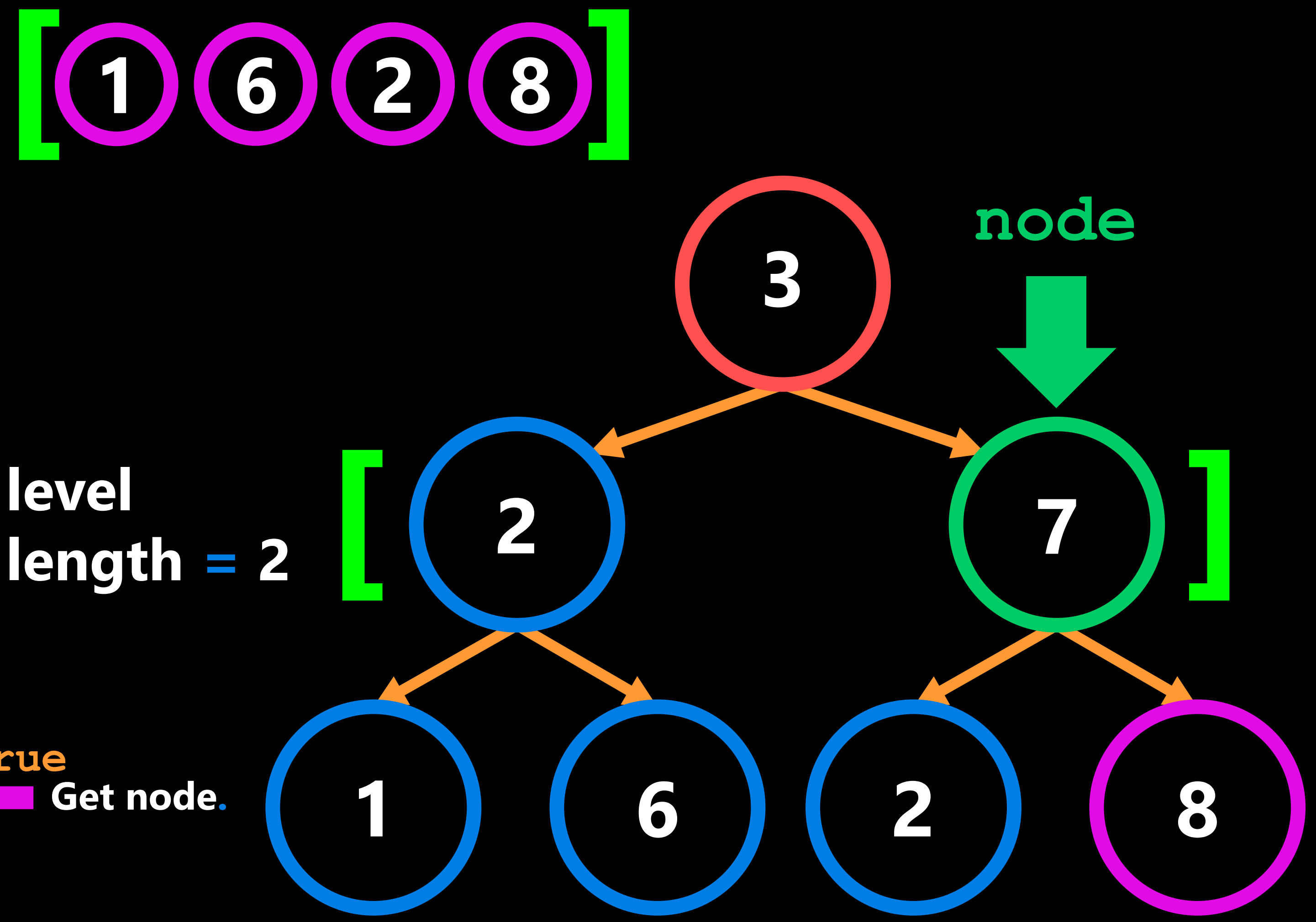
True ◀ if node.right is not None:

Get node. ◀ level_next.append(node.right)

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7
```

level_next =



level length = 2

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ (while len(level) > 0:)

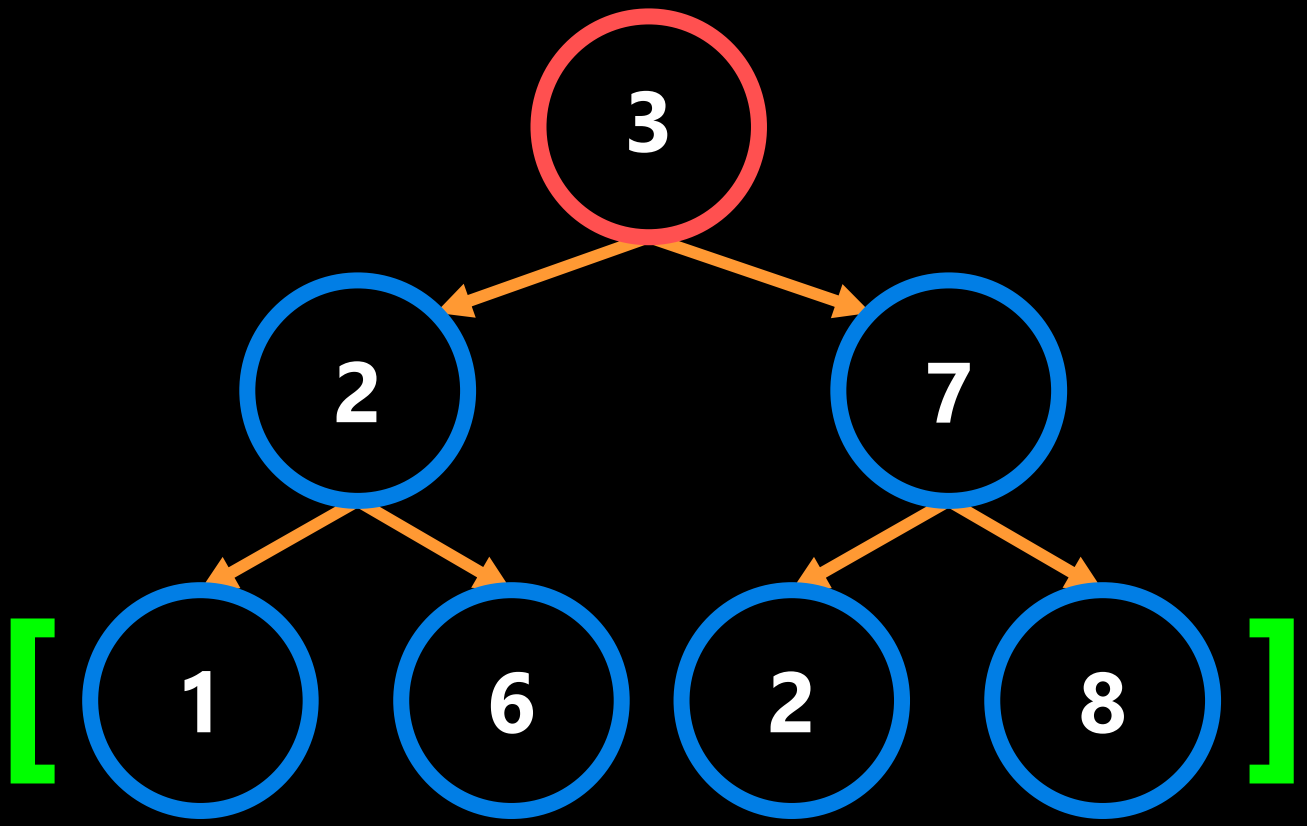
Move to next level.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7
```

level_next = [1 6 2 8]

level length = 4



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

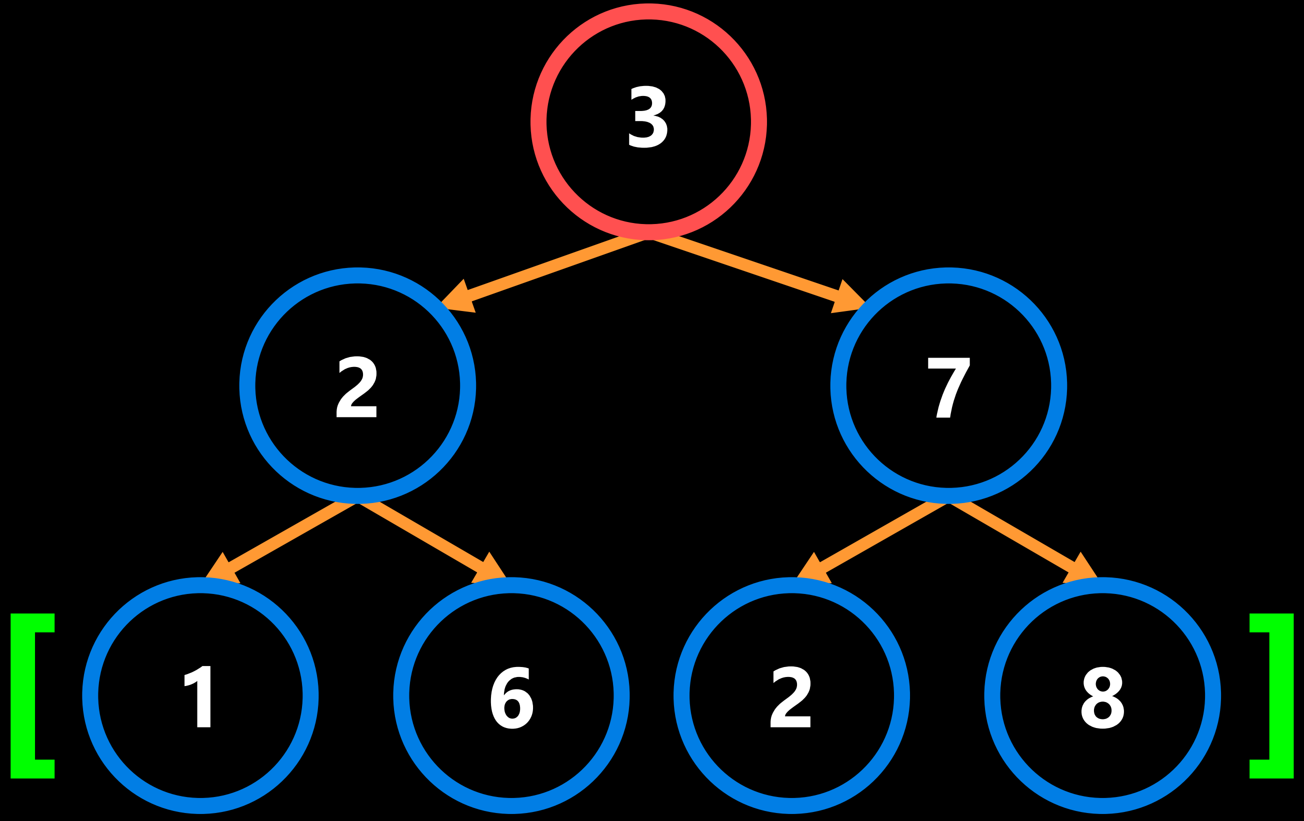
True ▶ while len(level) > 0:

Create empty list.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7
```

level_next = [ ]



level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

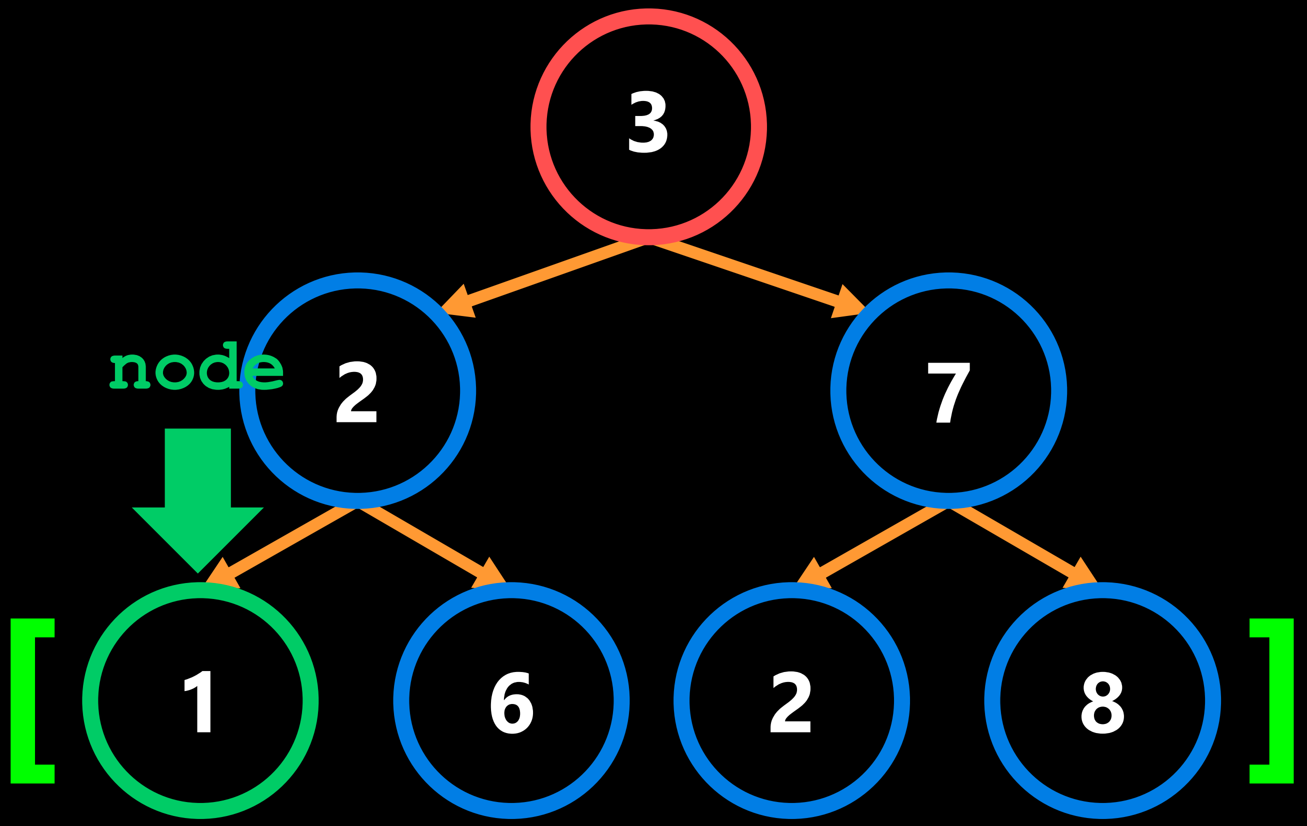
Loop through nodes in level.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7
```

level_next = [ ]

level length = 4



```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

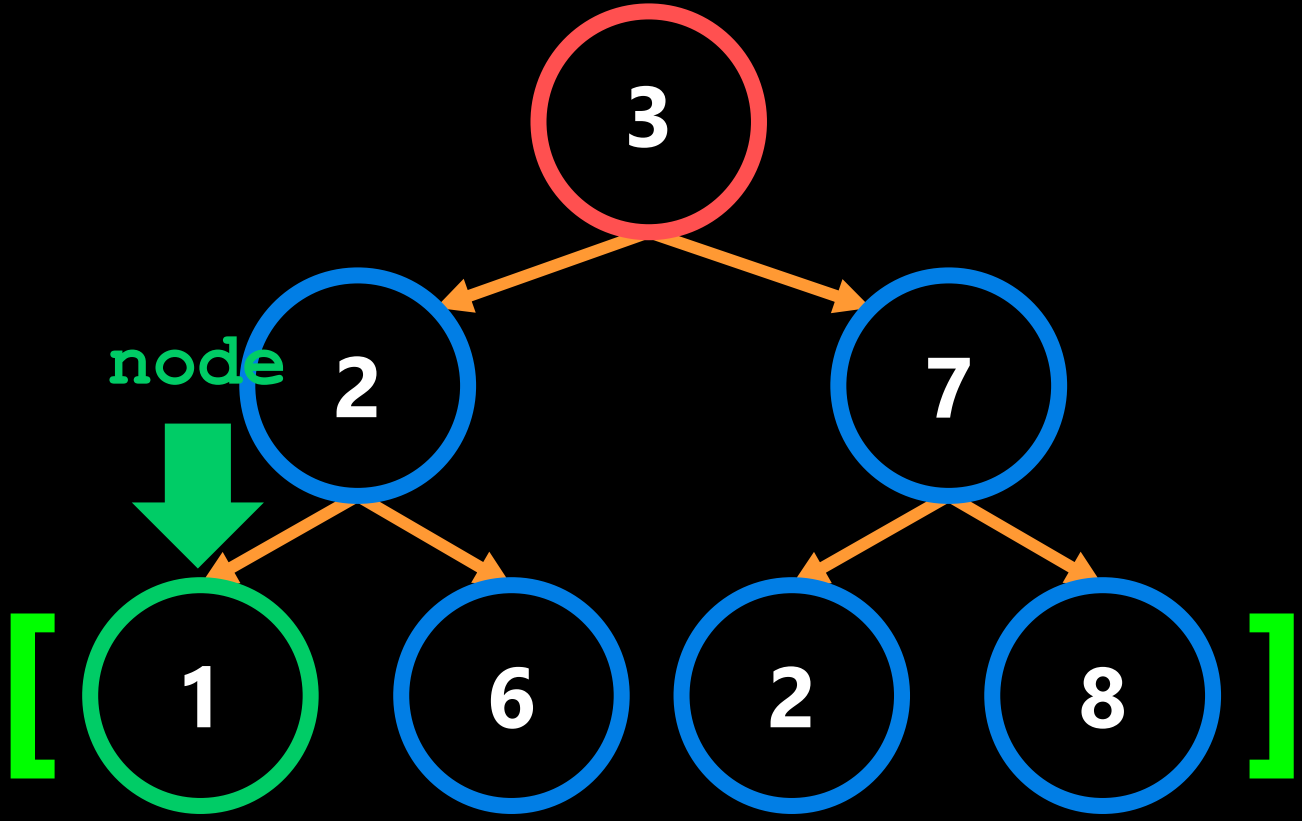
True

print.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1
```

level_next = [ ]



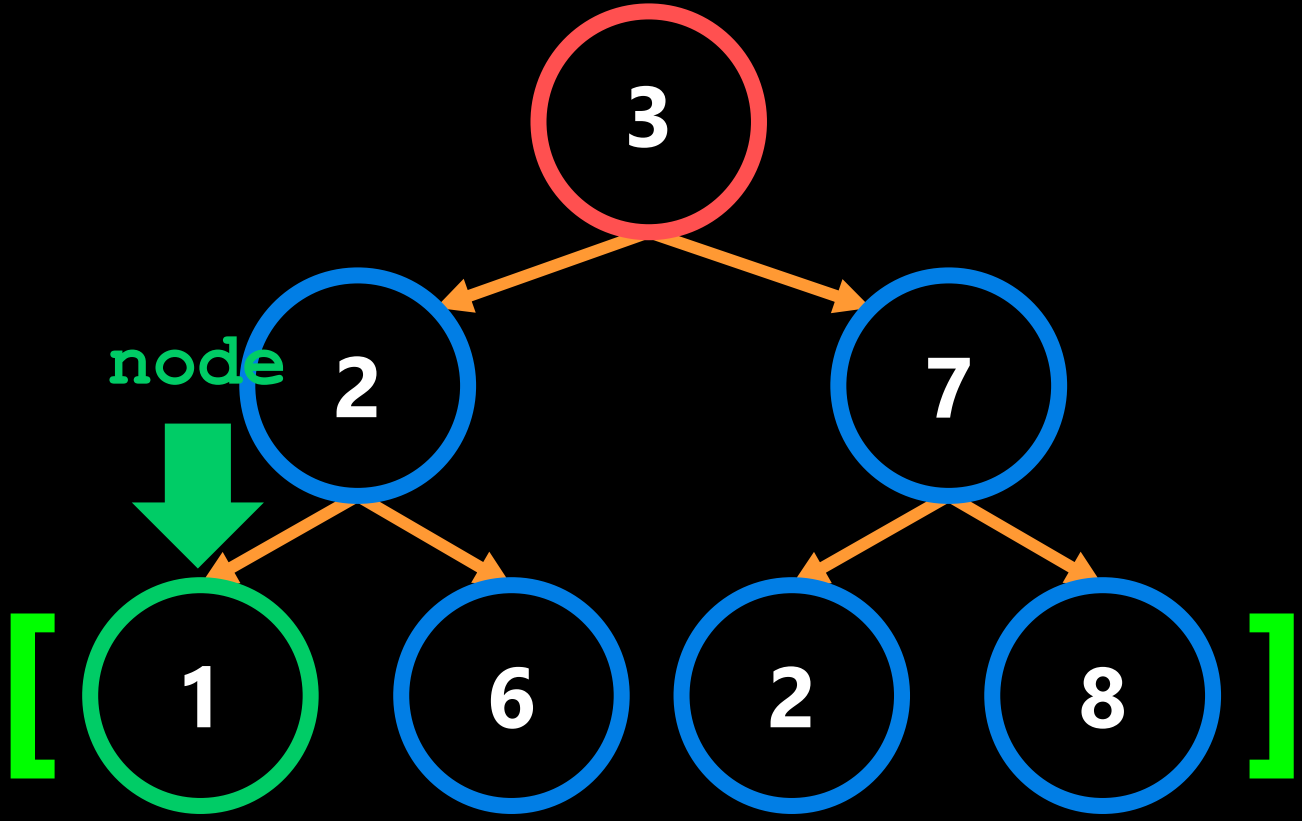
level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

False ▶         if node.left is not None:
                    level_next.append(node.left)
False ▶         if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1
```

level_next = [ ]



level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

True ▶ while len(level) > 0:

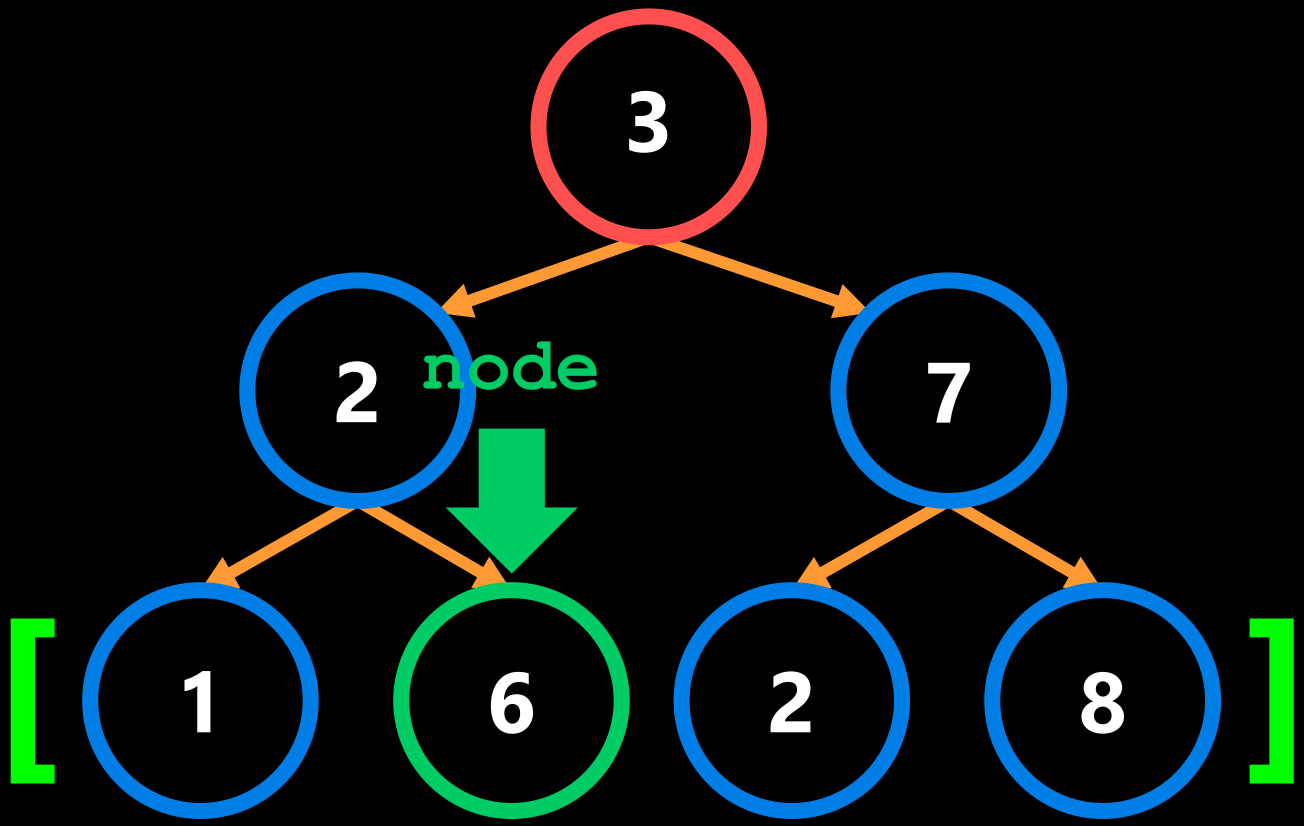
Loop through nodes in level.

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1
```

level_next = [ ]

level length = 4

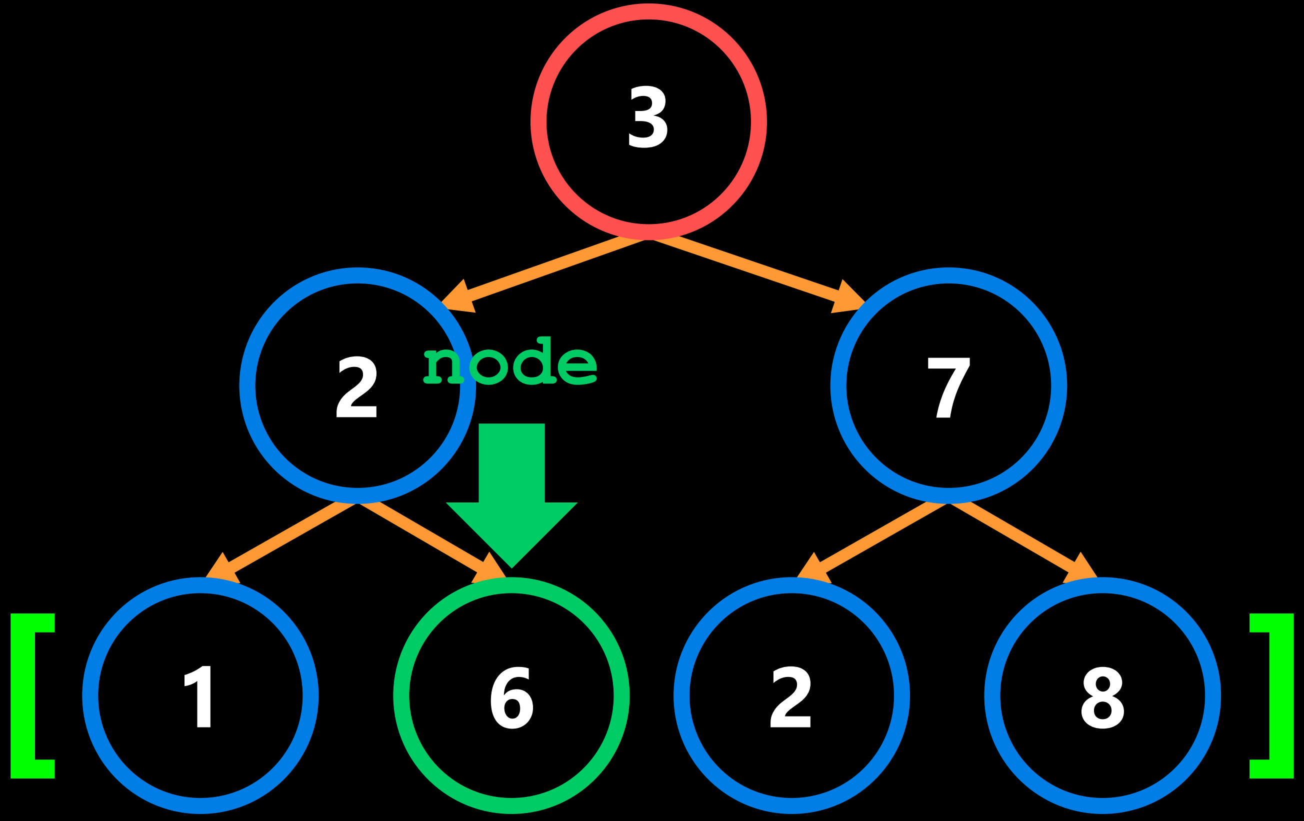


```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")  ◀ print.

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6
```

level_next = [ ]



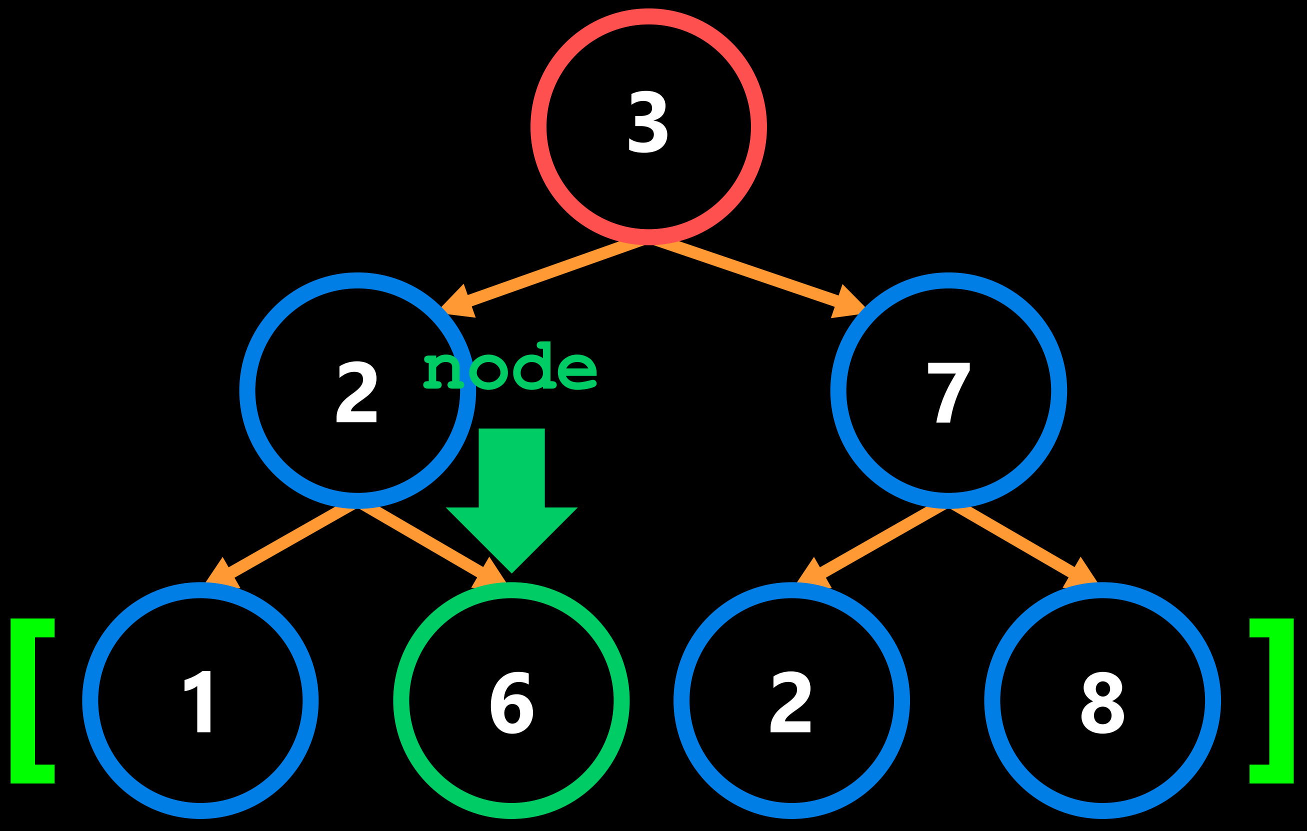
level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

False ▶         if node.left is not None:
                    level_next.append(node.left)
False ▶         if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6
```

level_next = [ ]



level

length = 4

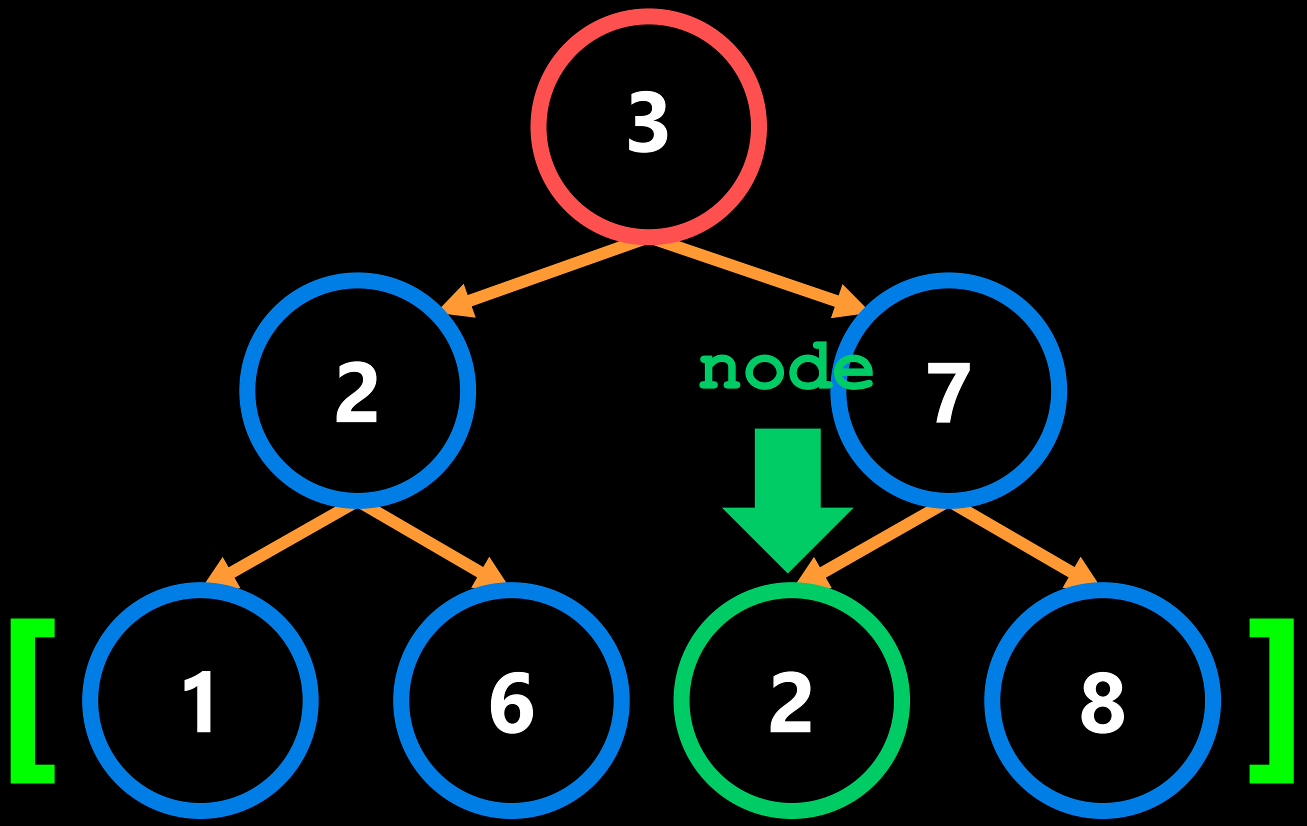
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:  ◀━━━

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

Loop through nodes in level.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6
```

level_next = [ ]



level length = 4

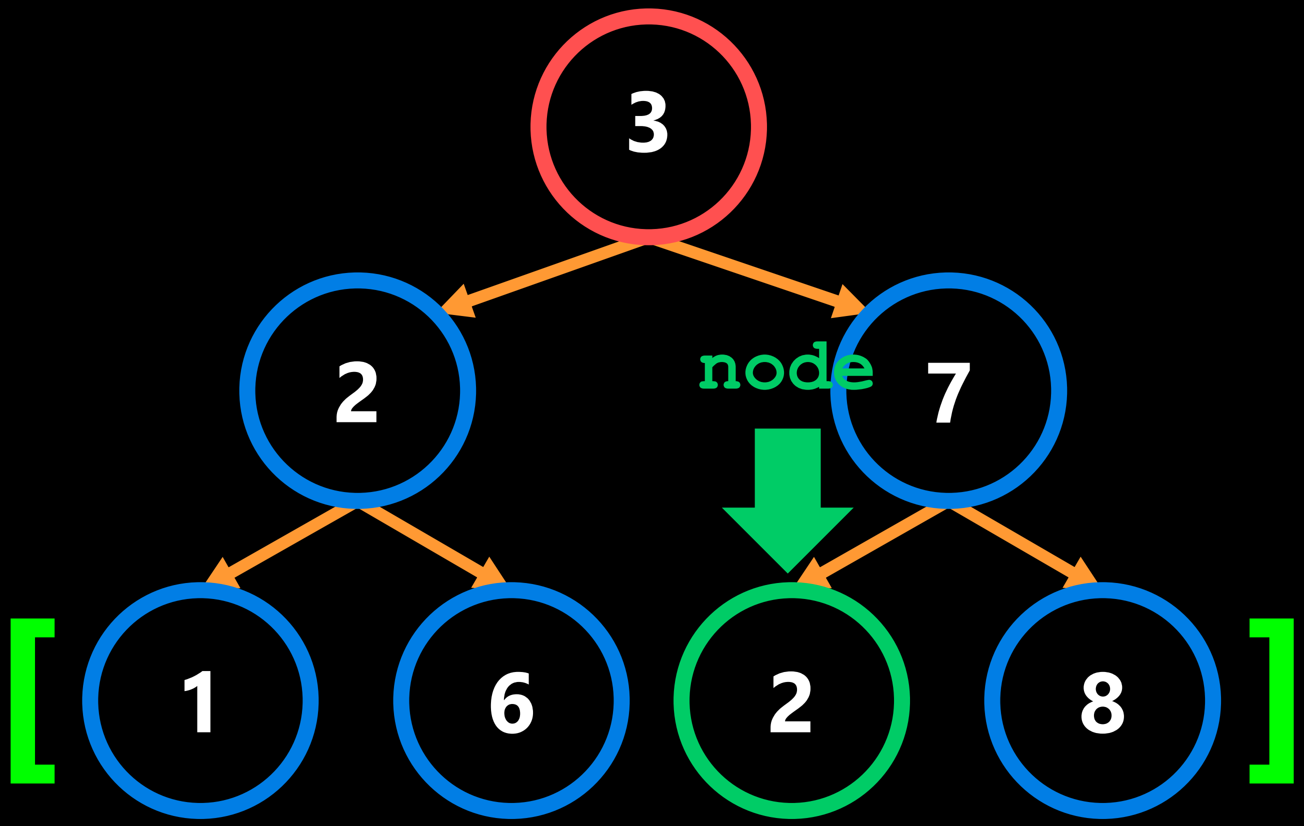
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="") ◀ print.

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2
```

level_next = [ ]

level length = 4

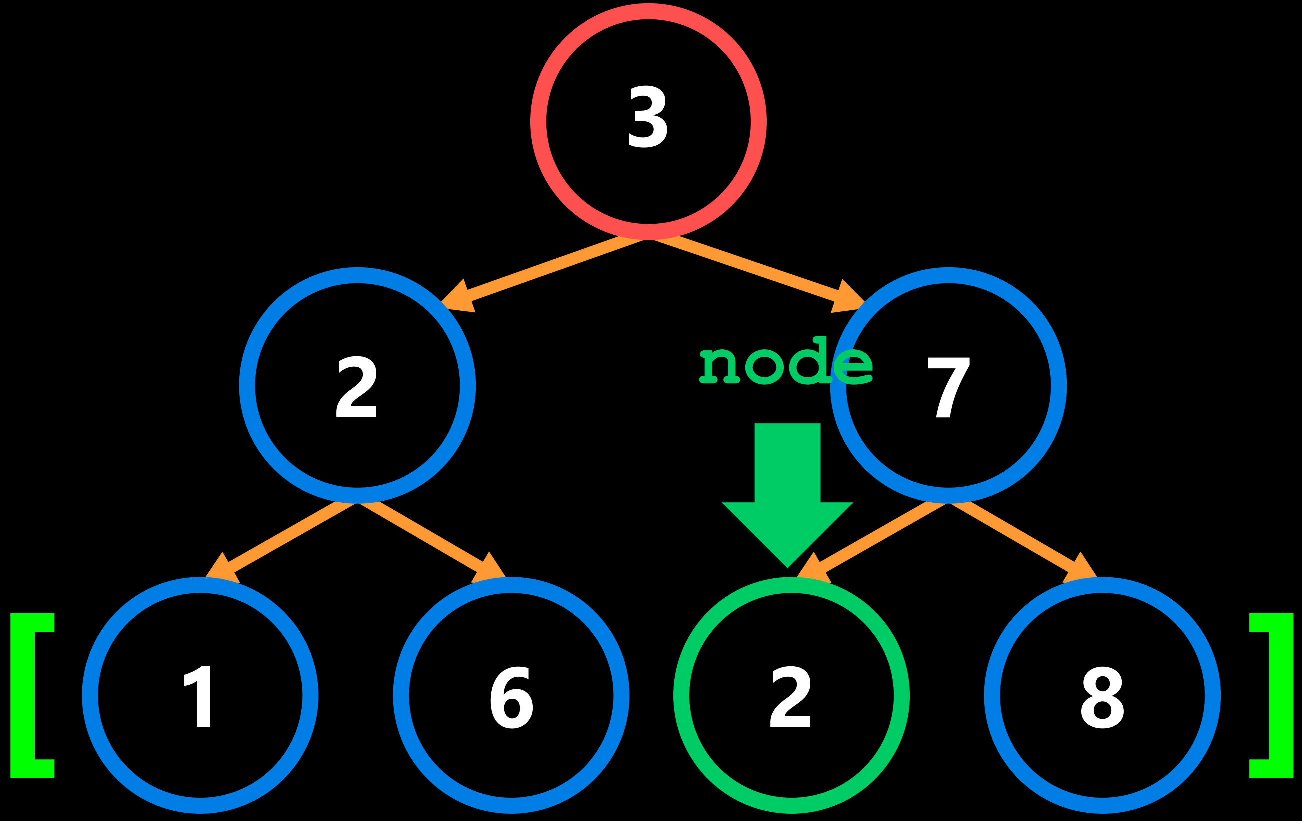


```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]
True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

    False ➡ if node.left is not None:
                    level_next.append(node.left)
    False ➡ if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2
```

level_next = [ ]



level length = 4

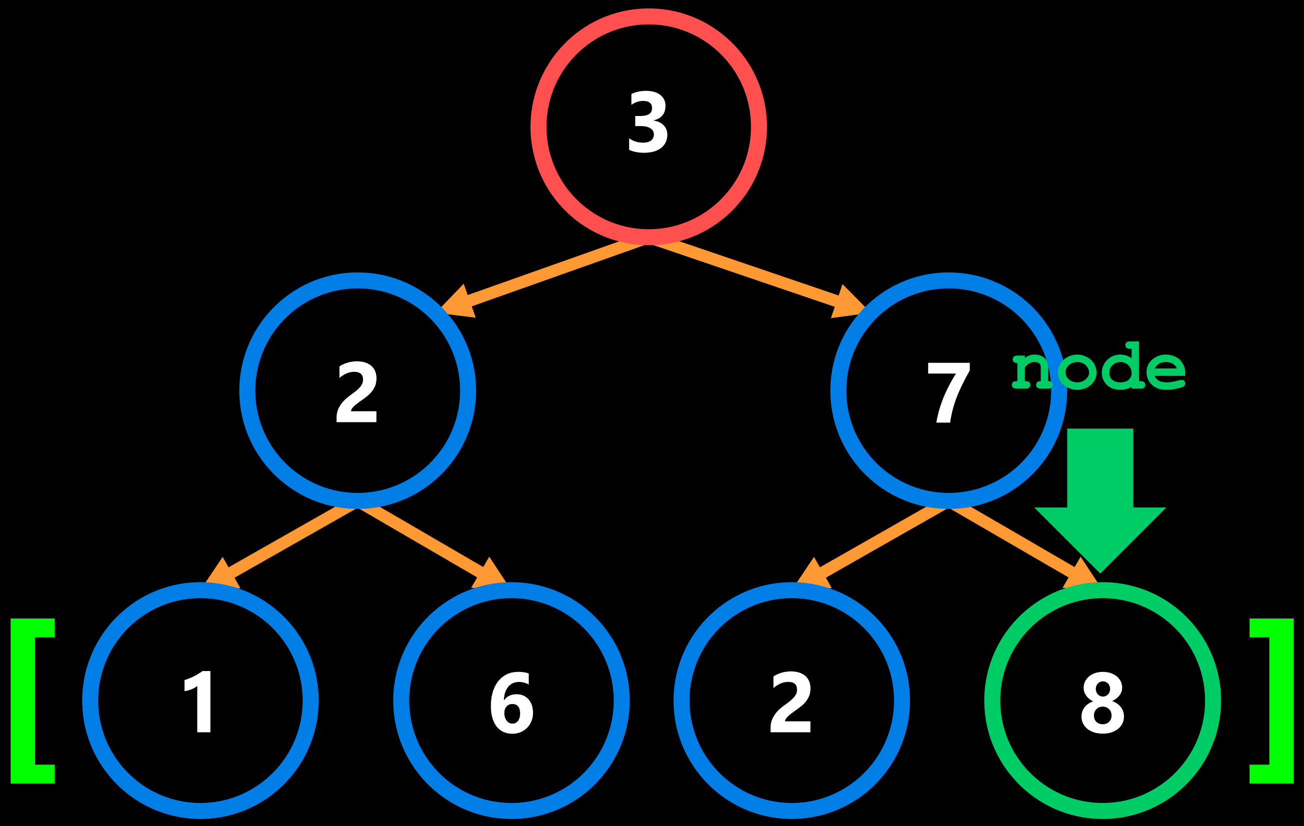
```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

Loop through nodes in level.

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2
```

level_next = [ ]



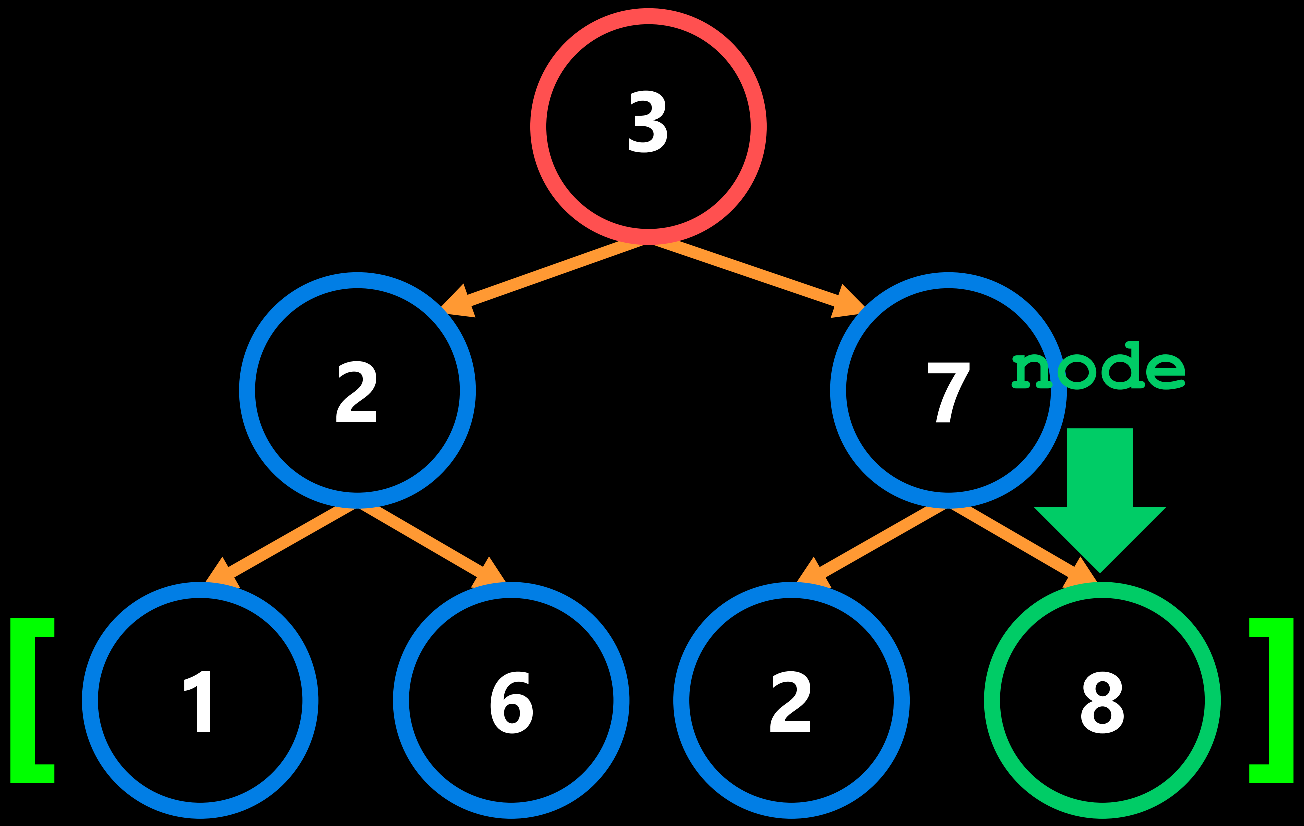
level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")  ◀ print.

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2  8
```

level_next = [ ]



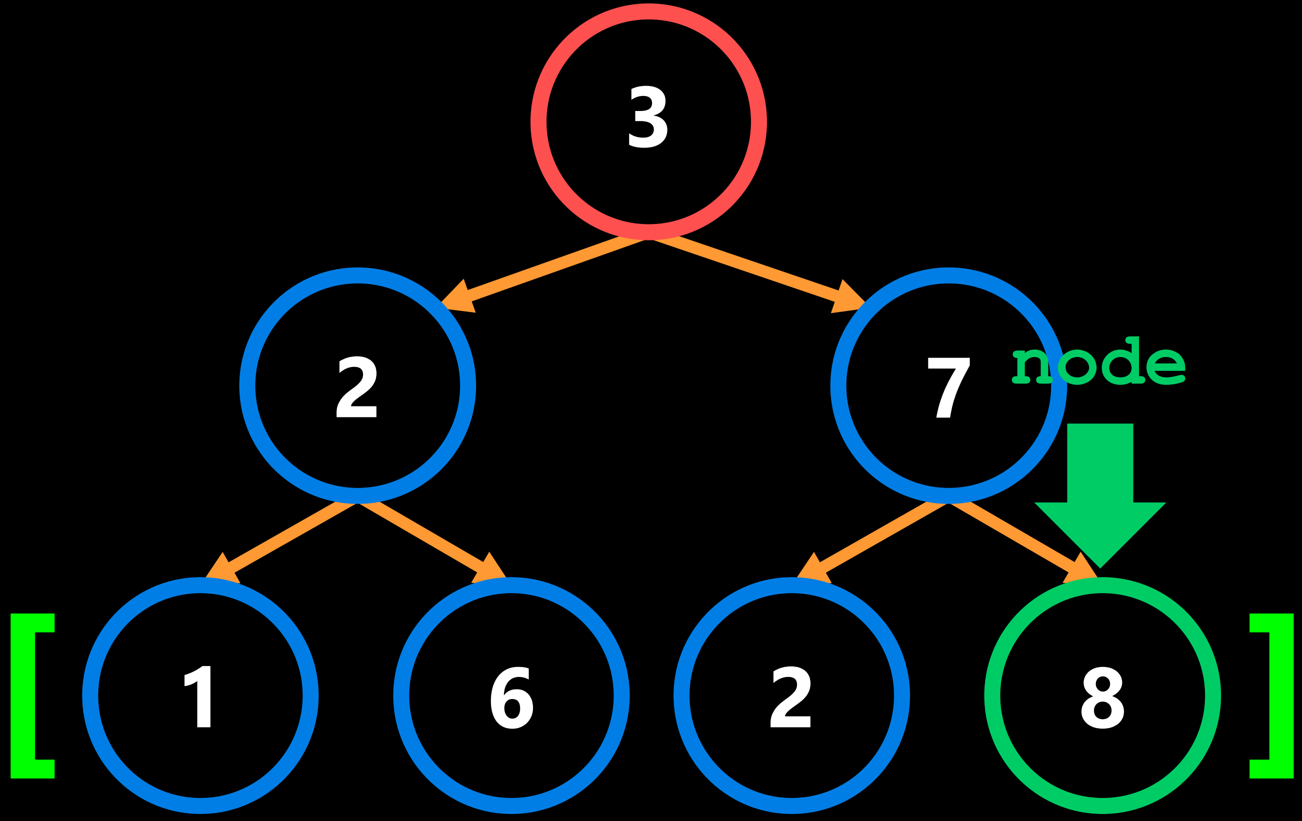
level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

False ➡     if node.left is not None:
                    level_next.append(node.left)
False ➡     if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2  8
```

level_next = [ ]



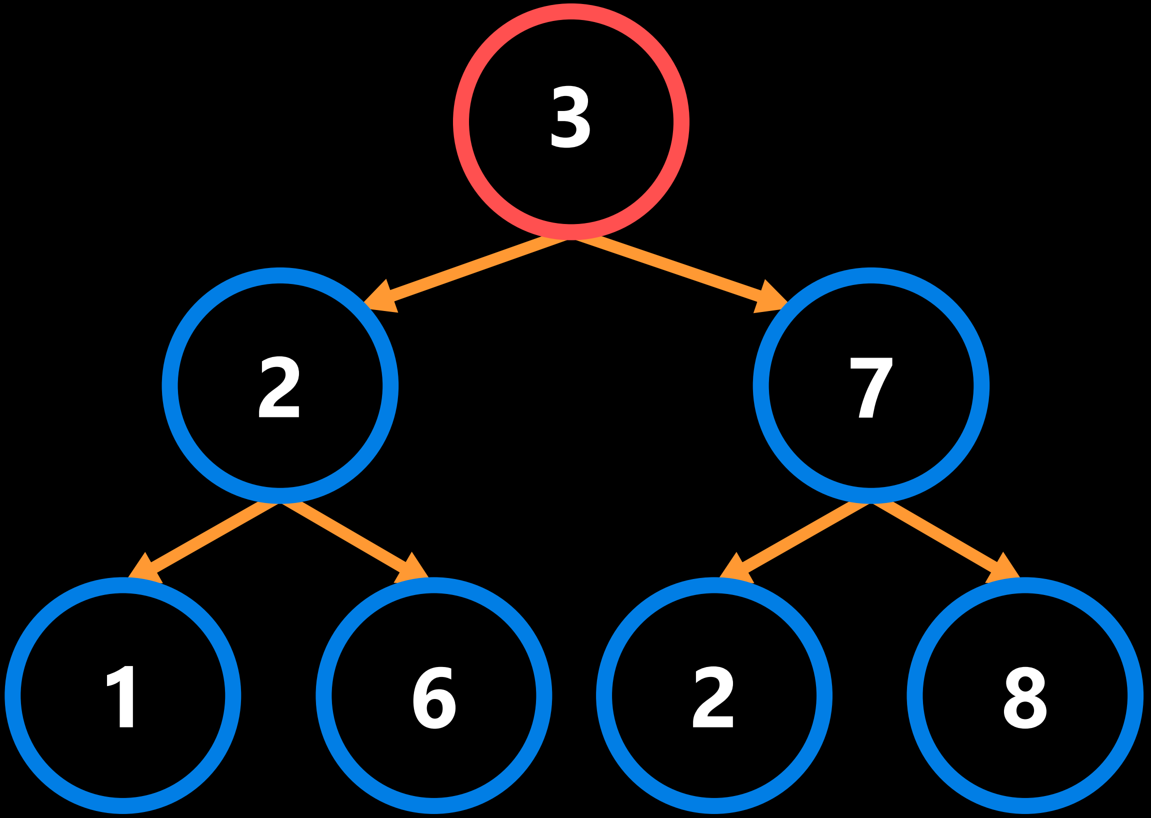
level length = 4

```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

True ▶  while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next   ◀ Move to next level.
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2  8
```

level = [ ]   length = 0

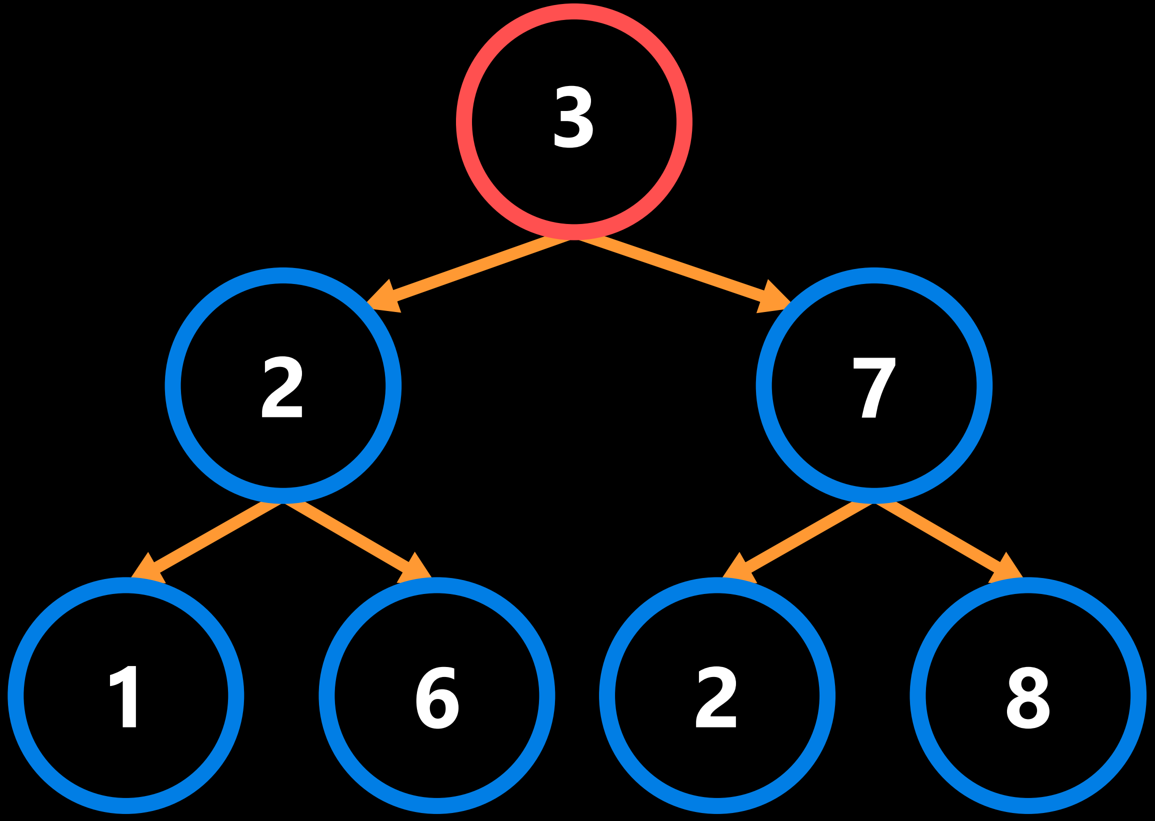


```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

False ▶ while len(level) > 0:

            level_next = []

            for node in level:

                print(node.cargo, " ", end="")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                           TreeNode(2, TreeNode(1), TreeNode(6)),
                           TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2  8
```

level = [ ]   length = 0

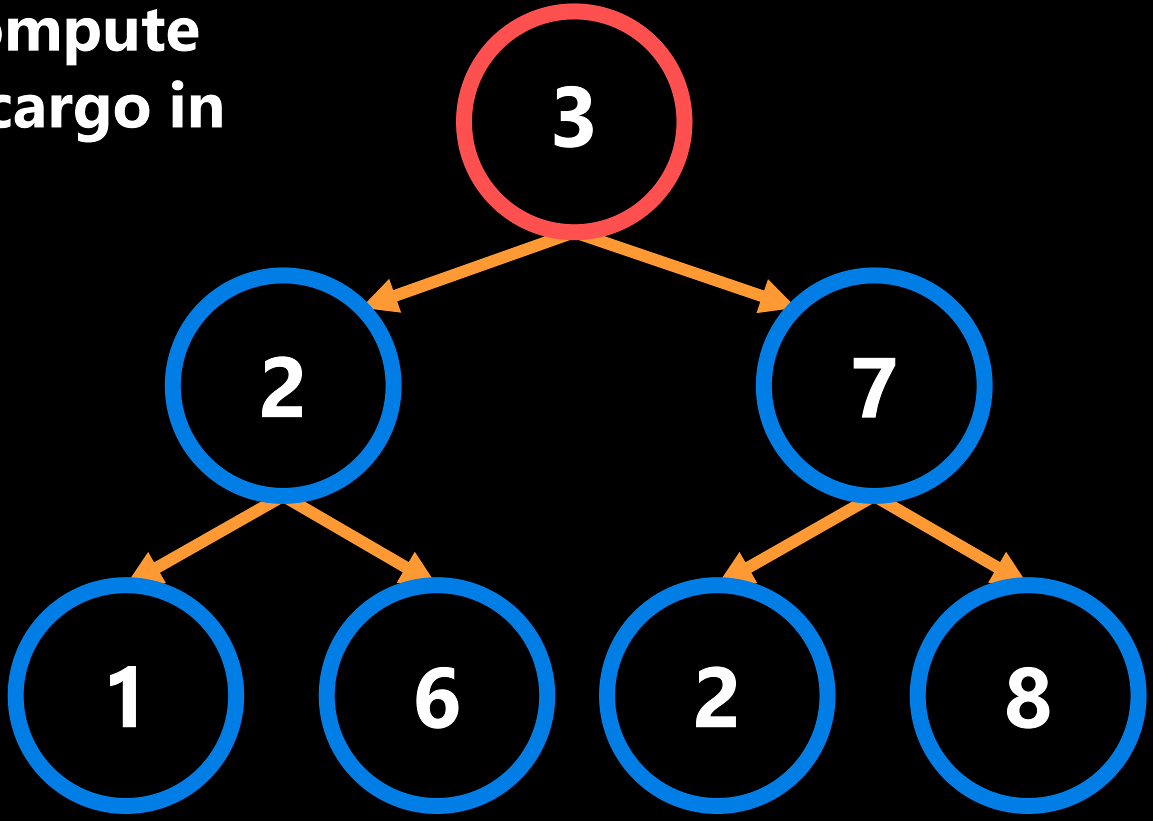


```
class BinaryTree:
    """A Node class used by a binary tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                cargo_sum += node.cargo

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

```
tree = BinaryTree(TreeNode(3,
                  TreeNode(2, TreeNode(1), TreeNode(6)),
                  TreeNode(7, TreeNode(2), TreeNode(8))))
```

```
tree.print_tree()
3

2  7

1  6  2  8
```

Cargo Sum: You can imagine using the same approach to compute the sum of all cargo in the tree.



APS106

UNIVERSITY OF TORONTO

linked lists and binary trees.

Week 12 | Lecture 1 (12.1)

if nothing else, write #cleancode