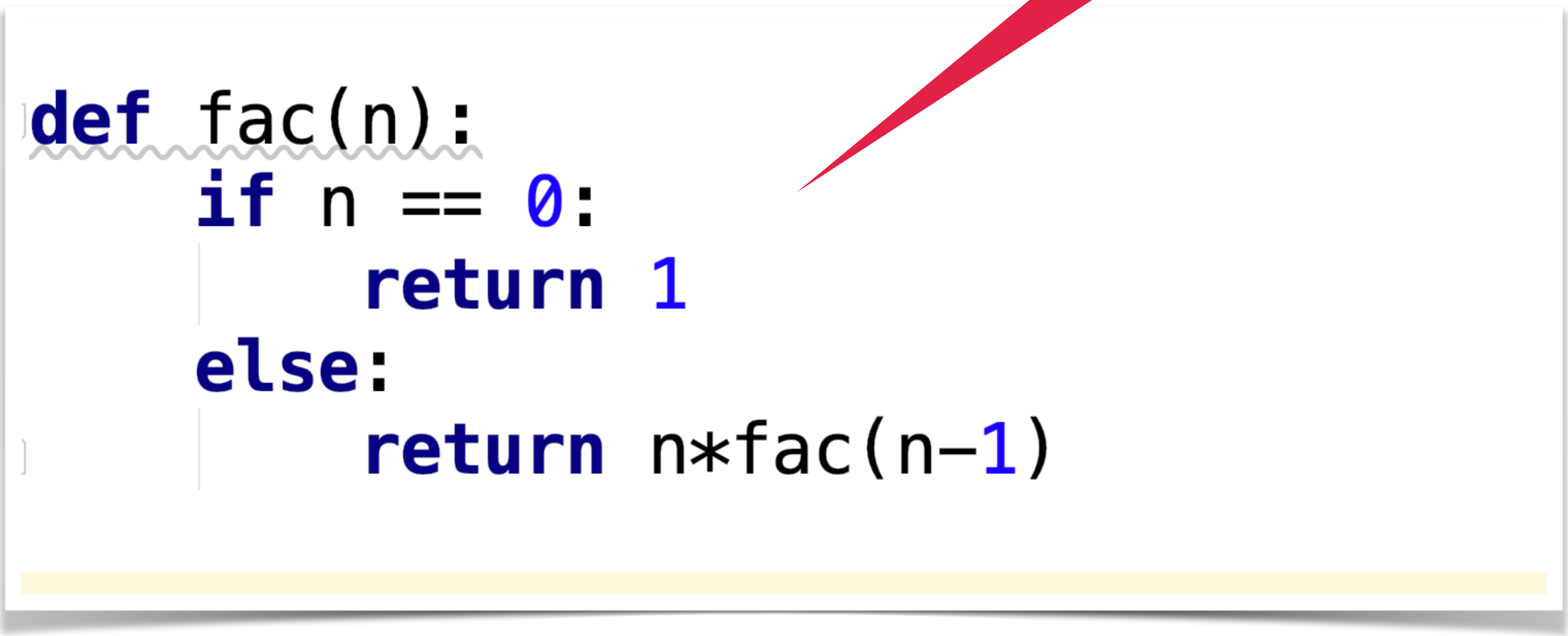
遞迴程式設計：
行列式與二元樹

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

問題大小n為初始問題，直接代入答案

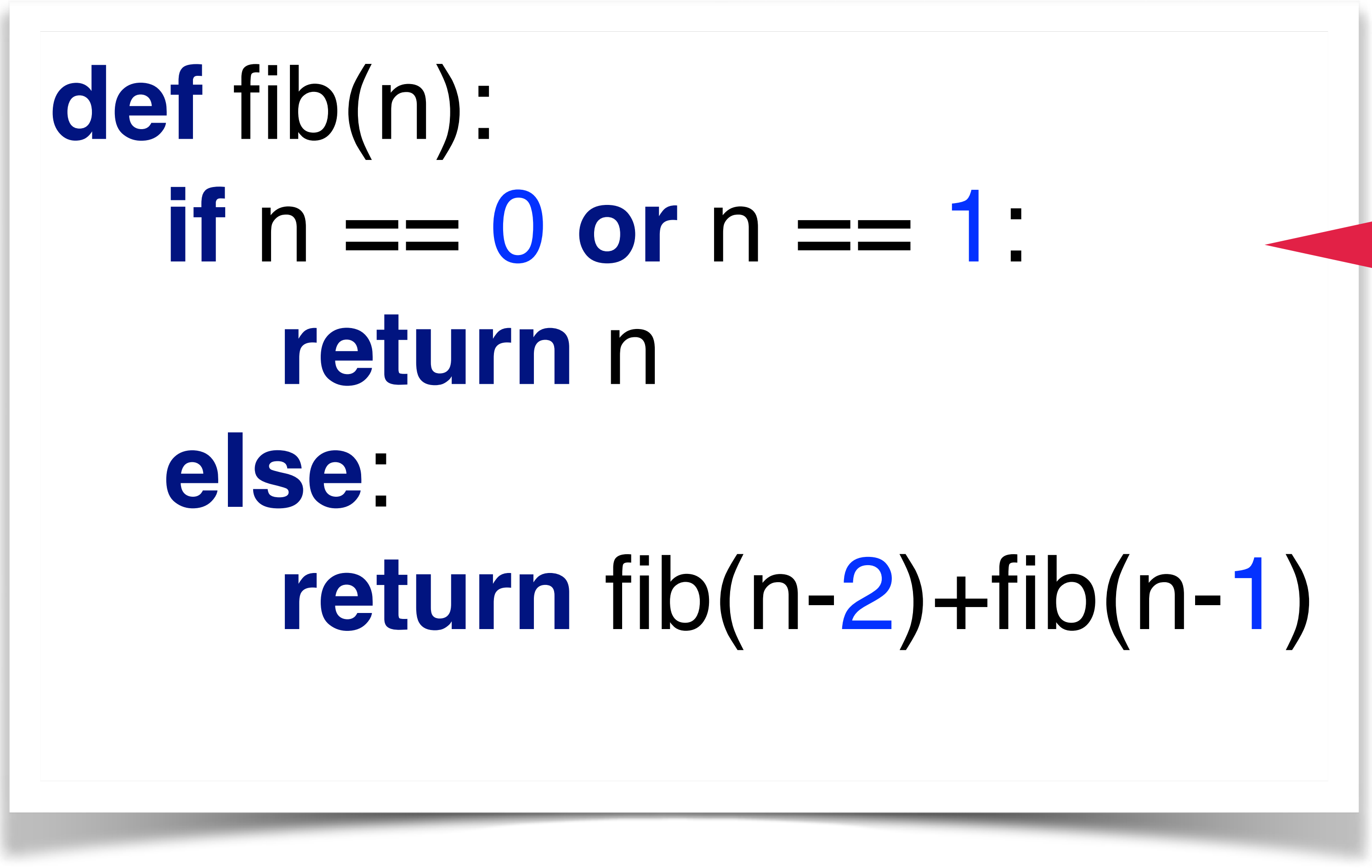


```
def fac(n):
    if n == 0:
        return 1
    else:
        return n*fac(n-1)
```

Fibonacci numbers

$$F_0 = 0, \quad F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \quad \mathit{if} \quad n > 1$$



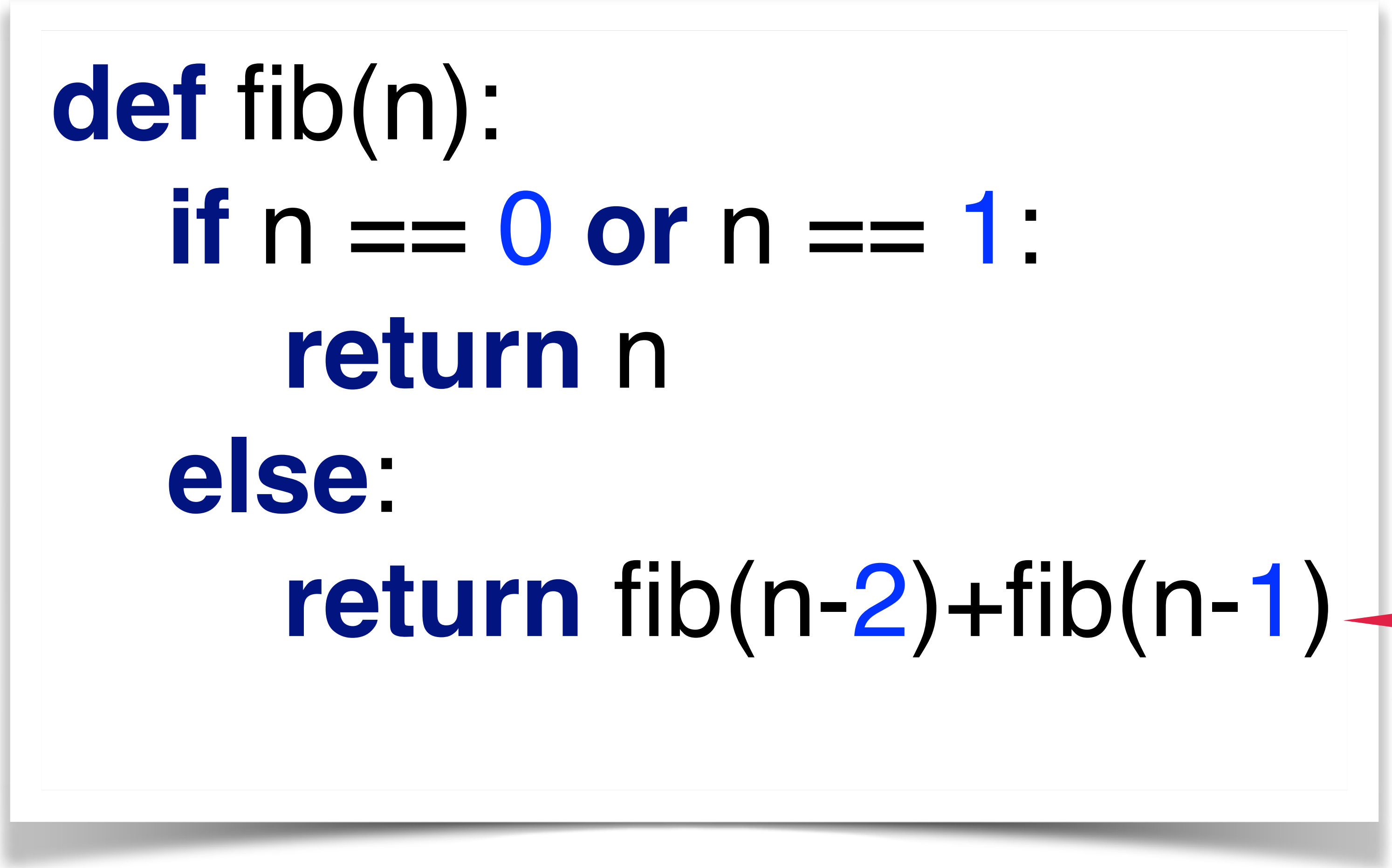
```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n-2)+fib(n-1)
```

問題大小n為初始問題，直接代入答案

Fibonacci numbers

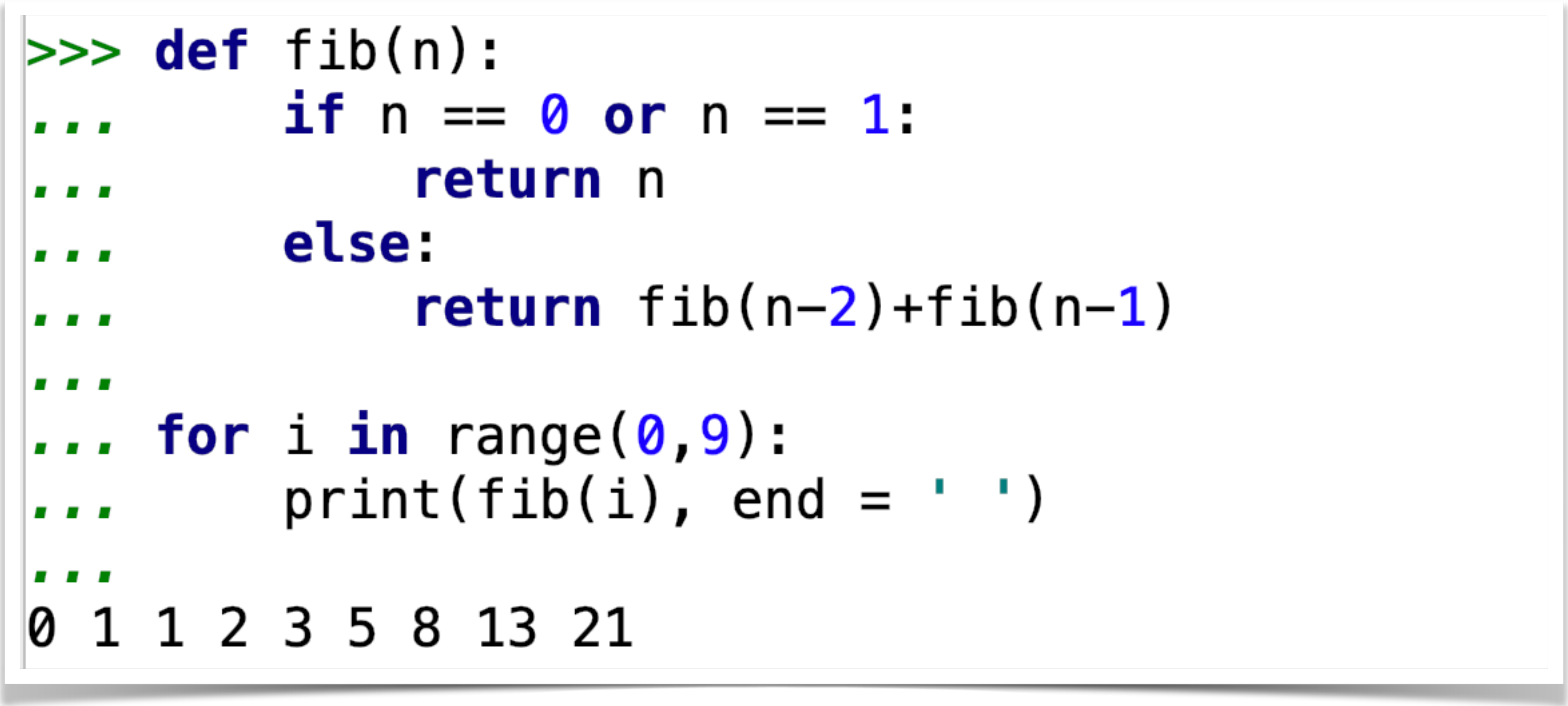
$$F_0 = 0, \quad F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \quad if \quad n > 1$$



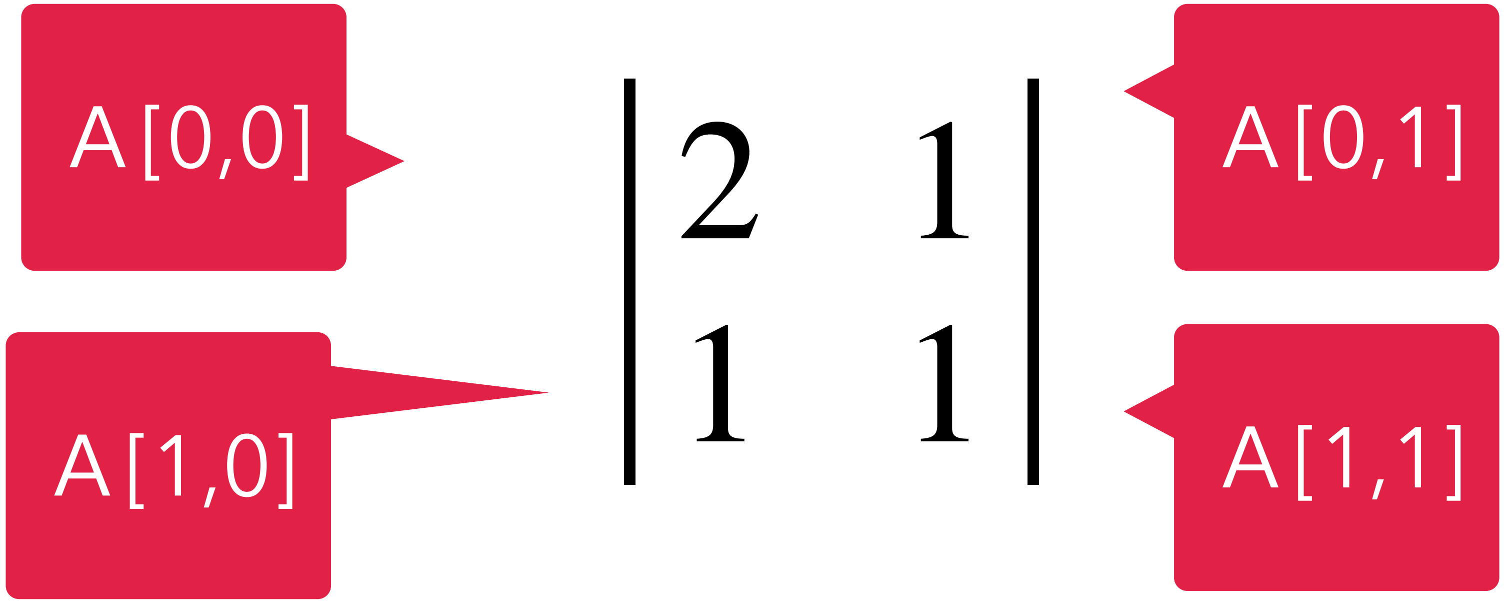
```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n-2)+fib(n-1)
```

問題大小n不是初始問題，使用前兩階的答案，合成問題大小n的答案表示式



```
>>> def fib(n):
...     if n == 0 or n == 1:
...         return n
...     else:
...         return fib(n-2)+fib(n-1)
...
... for i in range(0,9):
...     print(fib(i), end = ' ')
...
0 1 1 2 3 5 8 13 21
```

Determinant



A[0,0]

A[0,1]

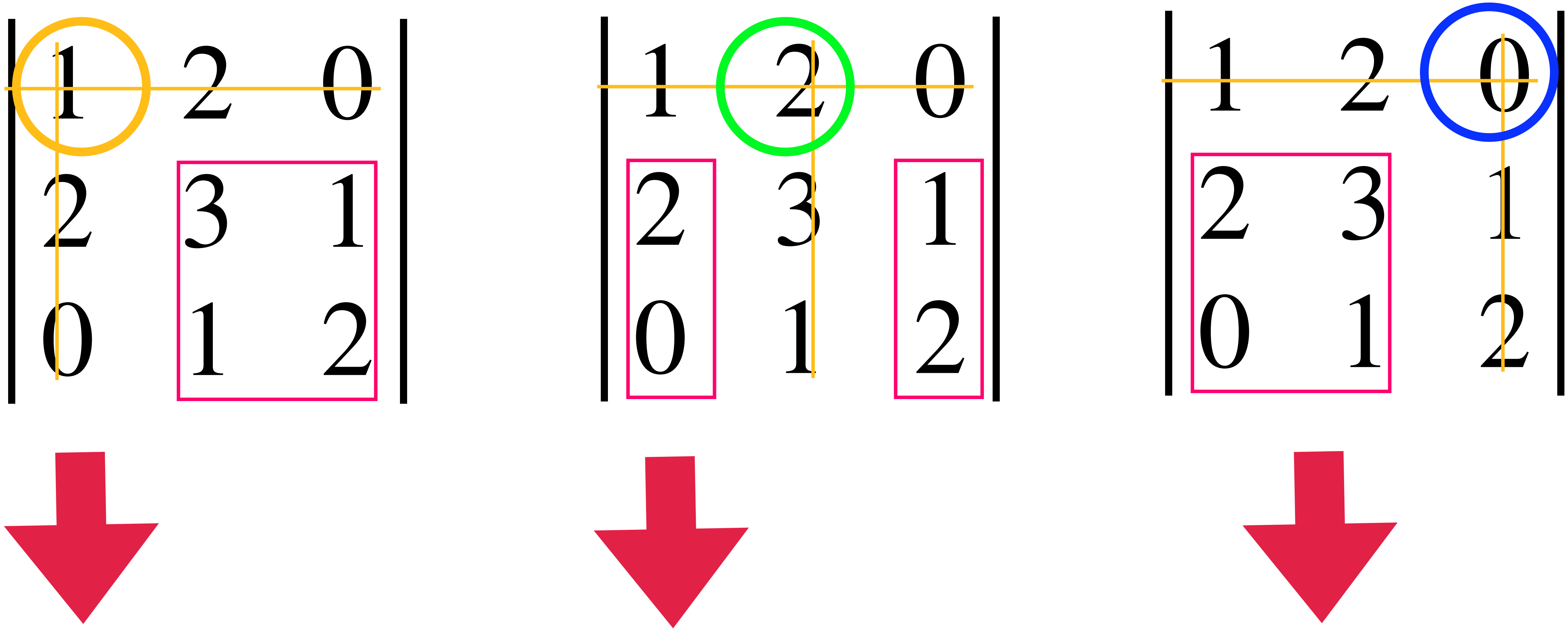
$$\begin{vmatrix} 2 & 1 \\ 1 & 1 \end{vmatrix}$$

A[1,0]

A[1,1]

```
def mydet(A):
    n,m = A.shape
    if n == 2 and m == 2:
        return A[0,0]*A[1,1]-A[0,1]*A[1,0]
```

問題大小n為初始問題，直接代入答案



$$ans = 1 \times \begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix} - 2 \times \begin{vmatrix} 2 & 1 \\ 0 & 2 \end{vmatrix} + 0 \times \begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix}$$

問題大小n不是初始問題，在等號右邊使用前一階問題的答案，合成問題大小n的答案表示式

$n = 3$

$i = 1$

$$\begin{array}{c|ccc|} 0 & 1 & 2 & 0 \\ 1 & 2 & 3 & 1 \\ 2 & 0 & 1 & 2 \end{array}$$

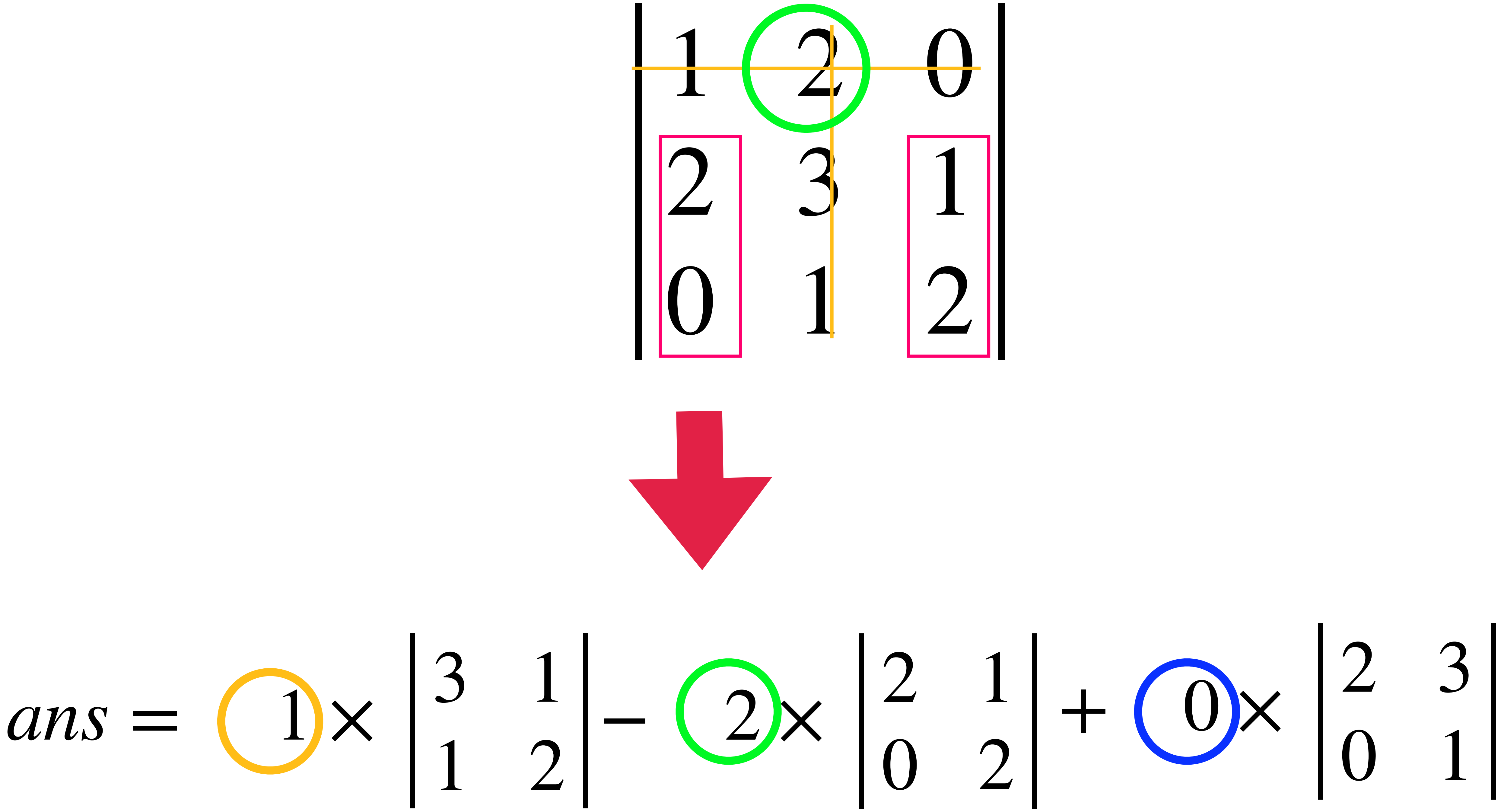
$$ans = 1 \times \begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix} - 2 \times \begin{vmatrix} 2 & 1 \\ 0 & 2 \end{vmatrix} + 0 \times \begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix}$$

```
B = np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
```

$$\begin{array}{cc} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{vmatrix} 2 & 1 & 0 & -1 \\ 2 & 3 & 1 & 2 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 0 & -1 \end{vmatrix} \end{array}$$

```
A = np.array([[2,1,0,-1],[2,3,1,2],[0,1,2,1],[1,2,0,-1]])
i = 2
n = 4
B1 = A[1:n,0:i]
B2 = A[1:n,i+1:n]
np.hstack((B1,B2))
array([[ 2,  3,  2],
       [ 0,  1,  1],
       [ 1,  2, -1]])
```

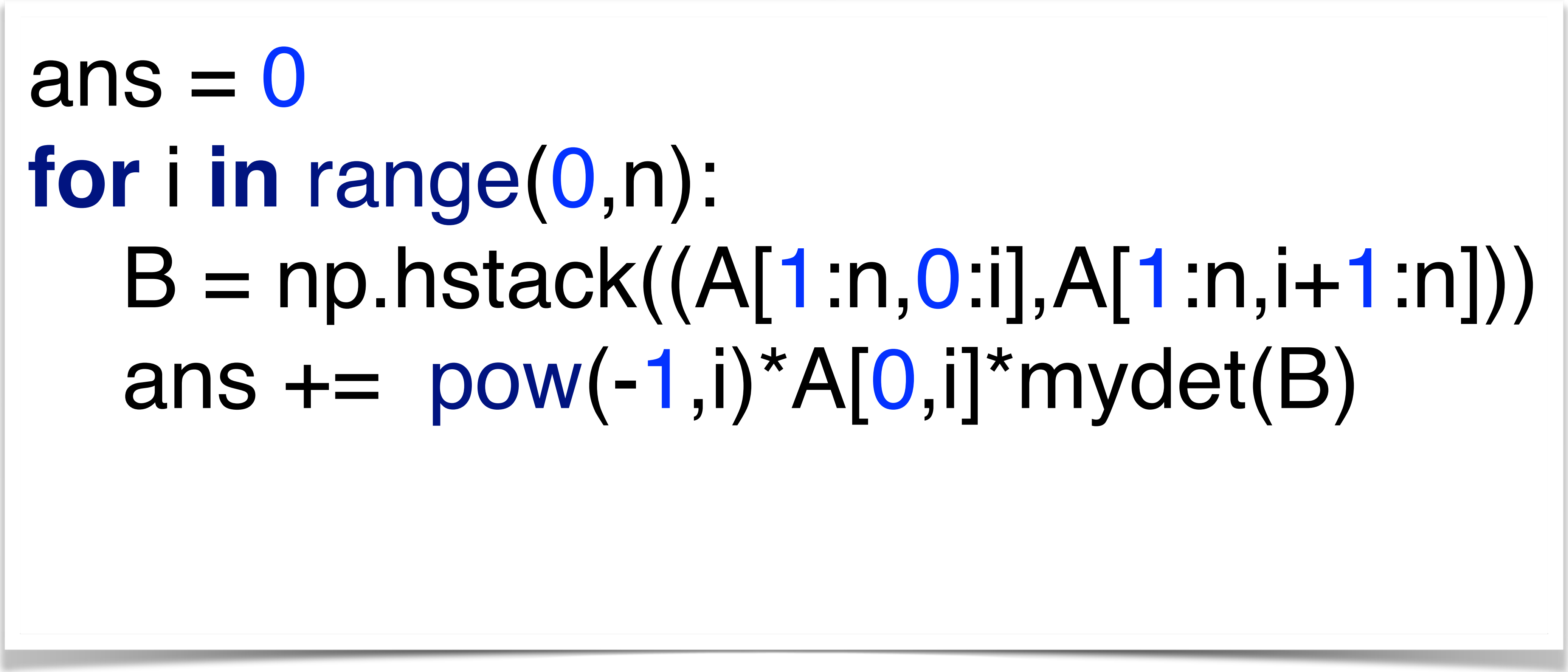
$$\begin{vmatrix} 2 & 3 & 2 \\ 0 & 1 & 1 \\ 1 & 2 & -1 \end{vmatrix}$$



$$\begin{vmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{vmatrix}$$

$$ans = 1 \times \begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix} - 2 \times \begin{vmatrix} 2 & 1 \\ 0 & 2 \end{vmatrix} + 0 \times \begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix}$$

```
B = np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
ans += pow(-1,i) * A[0, i] * mydet(B)
```



```
ans = 0
for i in range(0,n):
    B = np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
    ans +=  pow(-1,i)*A[0,i]*mydet(B)
```

$$ans = 1 \times \begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix} - 2 \times \begin{vmatrix} 2 & 1 \\ 0 & 2 \end{vmatrix} + 0 \times \begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix}$$

問題大小n為初始問題，直接代入答案

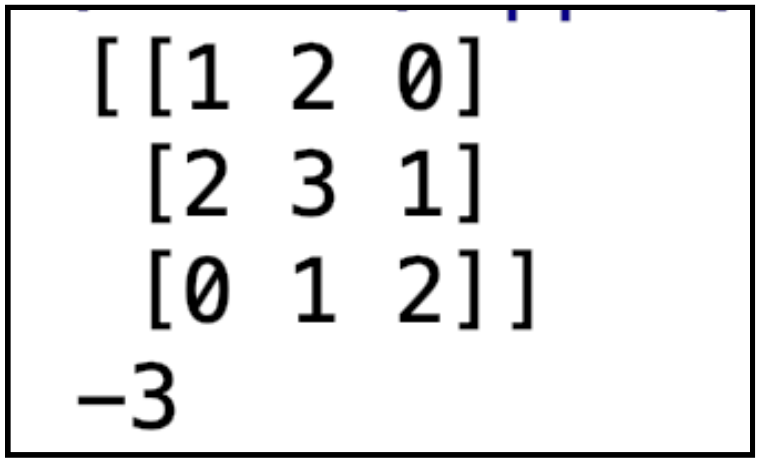
```
def mydet(A):
    n,m = A.shape
    if n == 2 and m == 2:
        return A[0,0]*A[1,1]-A[0,1]*A[1,0]
    else:
        ans = 0
        for i in range(0,n):
            B = np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
            ans +=  pow(-1,i)*A[0,i]*mydet(B)
        return ans
```

```
def mydet(A):
    n,m = A.shape
    if n == 2 and m == 2:
        return A[0,0]*A[1,1]-A[0,1]*A[1,0]
    else:
        ans = 0
        for i in range(0,n):
            B = np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
            ans +=  pow(-1,i)*A[0,i]*mydet(B)
        return ans
```

B 為(n-1)x(n-1)
矩陣

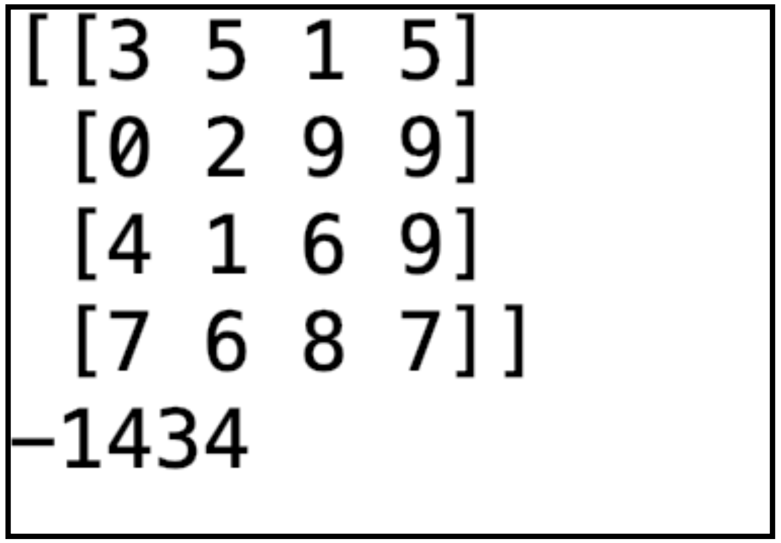
問題大小n不是初始問題，使用前一階問題的答案，合成問題大小n的答案表示式

```
import numpy as np
from mydet import mydet
A = np.matrix([[1,2,0],[2,3,1],[0, 1,2]])
print(A)
print(mydet(A))
```



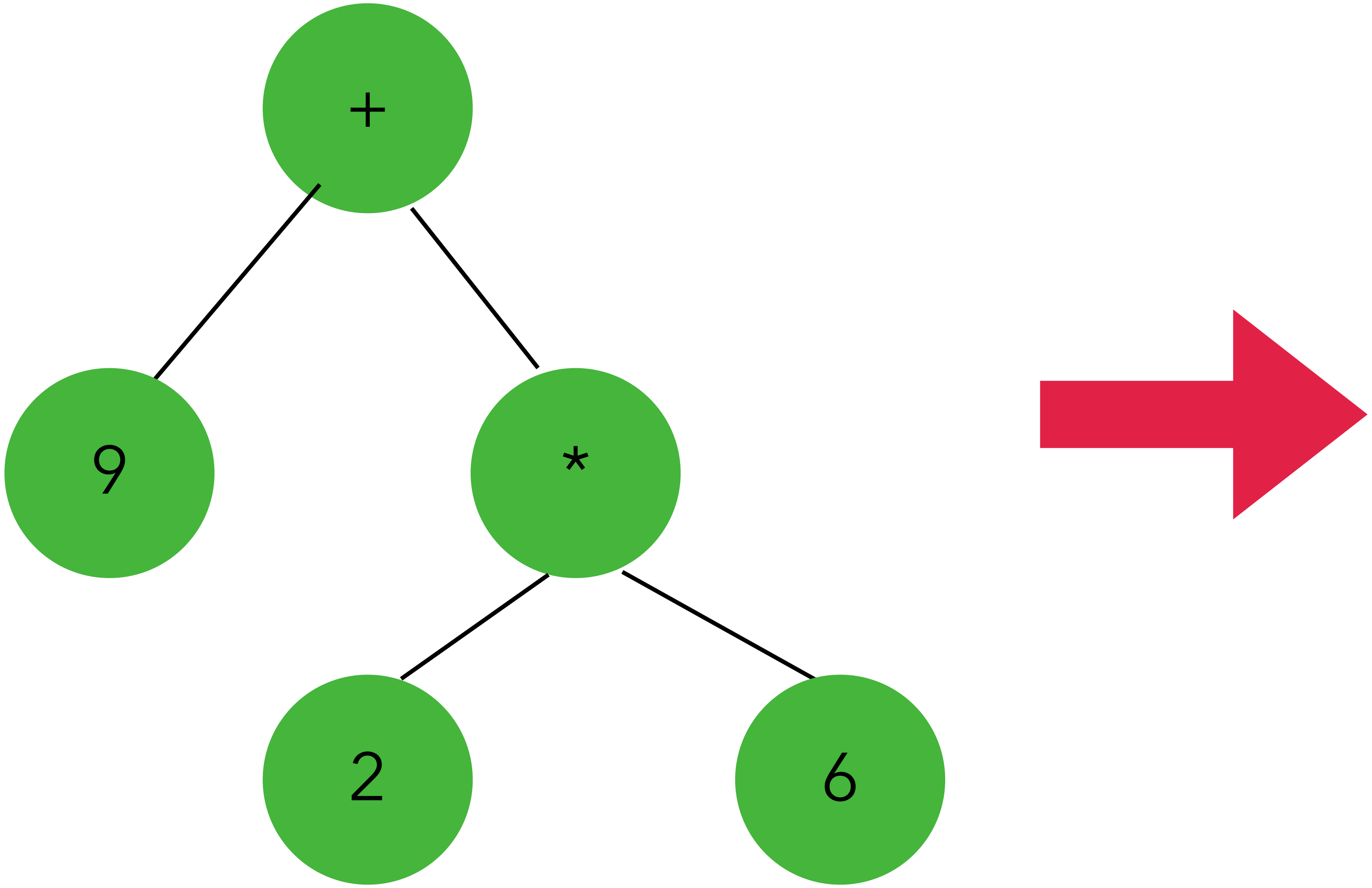
```
 [[1 2 0]
  [2 3 1]
  [0 1 2]]
-3
```

```
import numpy as np
from mydet import mydet

n =10
P = np.random.randint(0,10,(n,n))
print(P)
print(mydet(P))
```

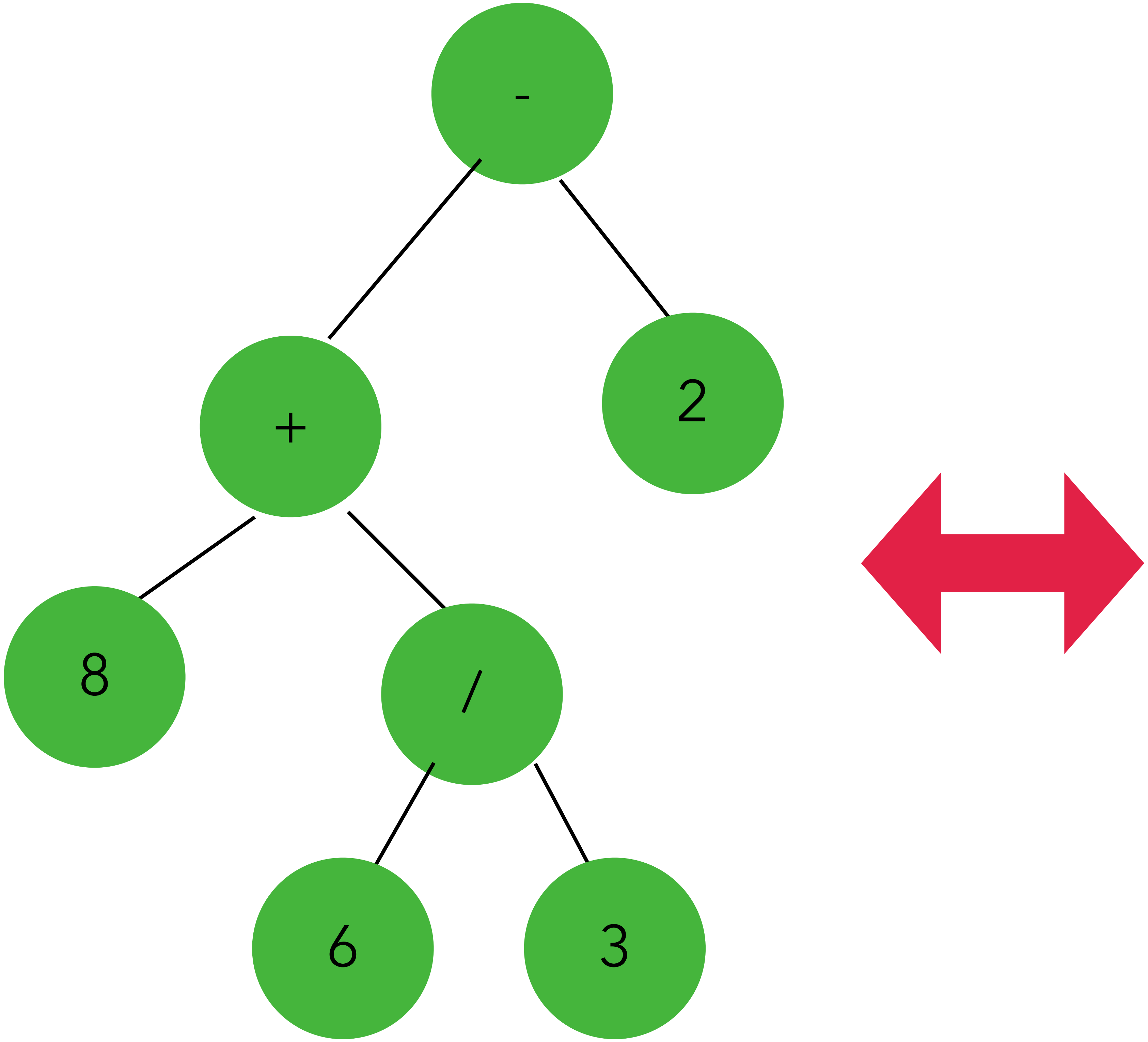


```
[[3 5 1 5]
 [0 2 9 9]
 [4 1 6 9]
 [7 6 8 7]]
-1434
```

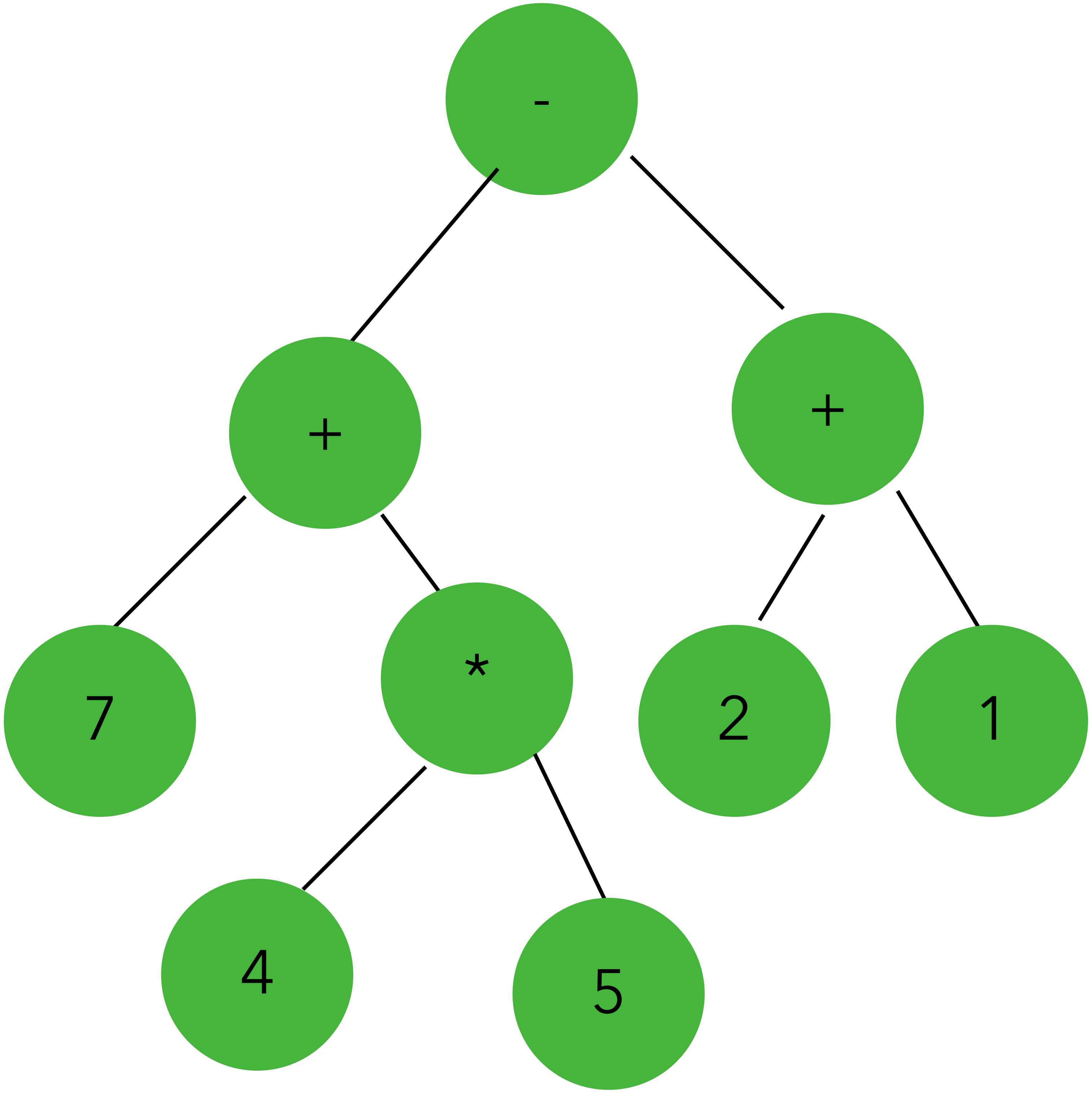
二元樹



9+2*6



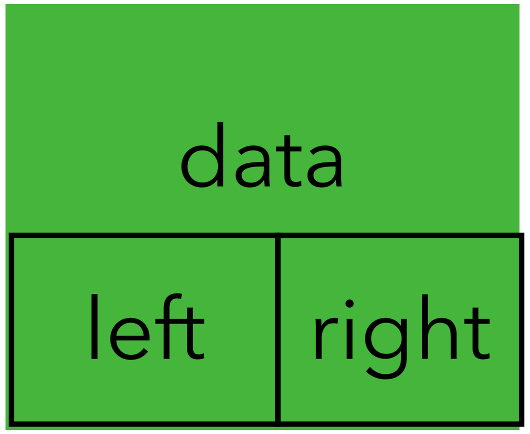
$8+6/3-2$



7+4∗5−(2+1)

如何建立節點？

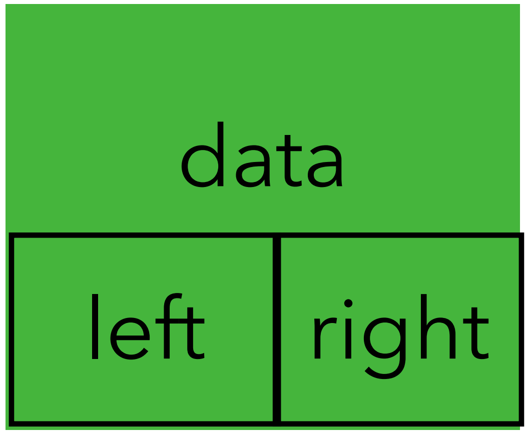
```
class Node:
    def __init__(self,data):
        self.data = data
        self.right = None
        self.left = None
```



data

left right

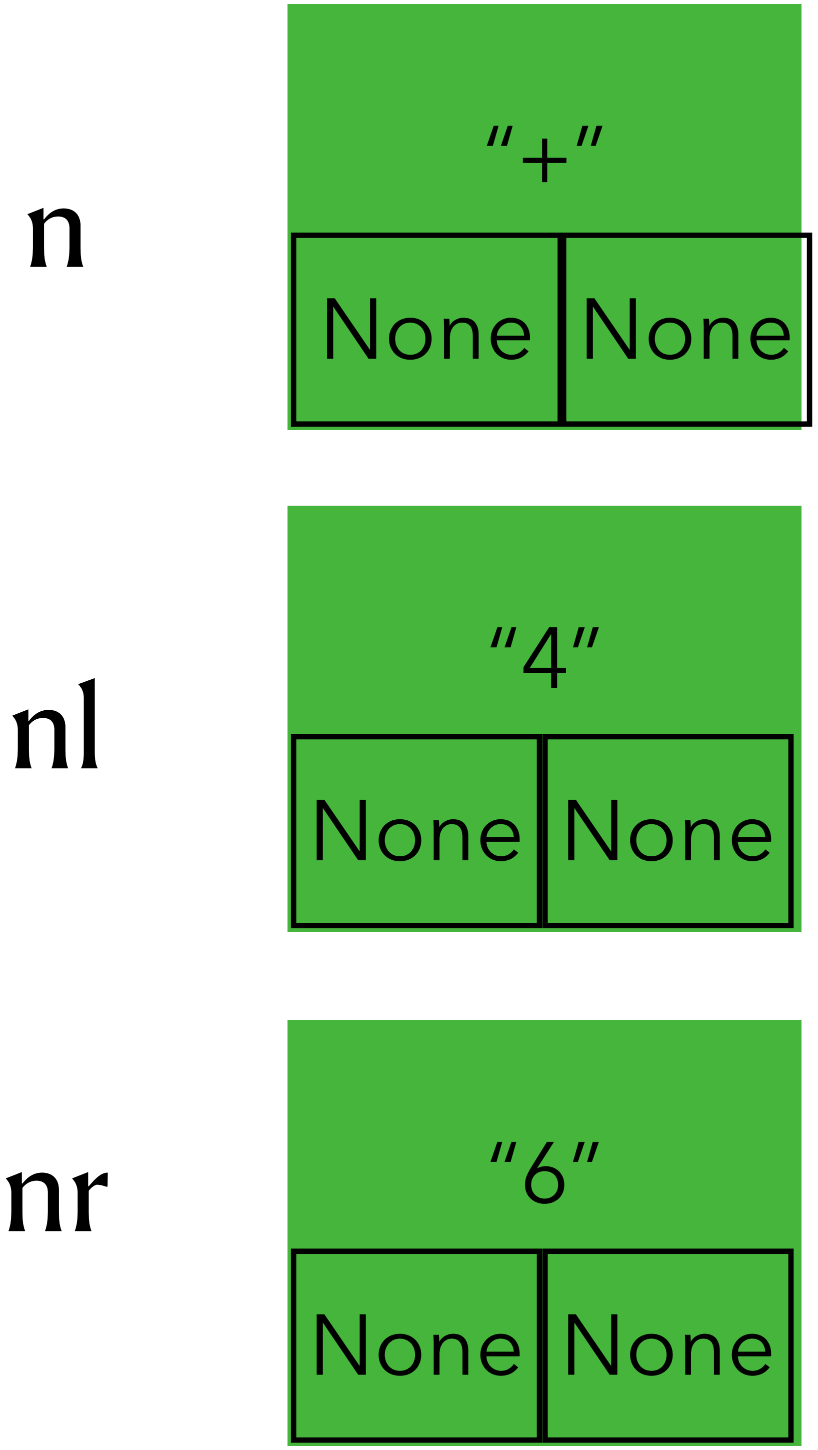
```
class Node:
    def __init__(self,data):
        self.data = data
        self.right = None
        self.left = None
```



data

left right

```
n = Node("+")
nl = Node("4")
nr = Node("6")
n.add(nl,nr)
```



n “+”

None None

nl “4”

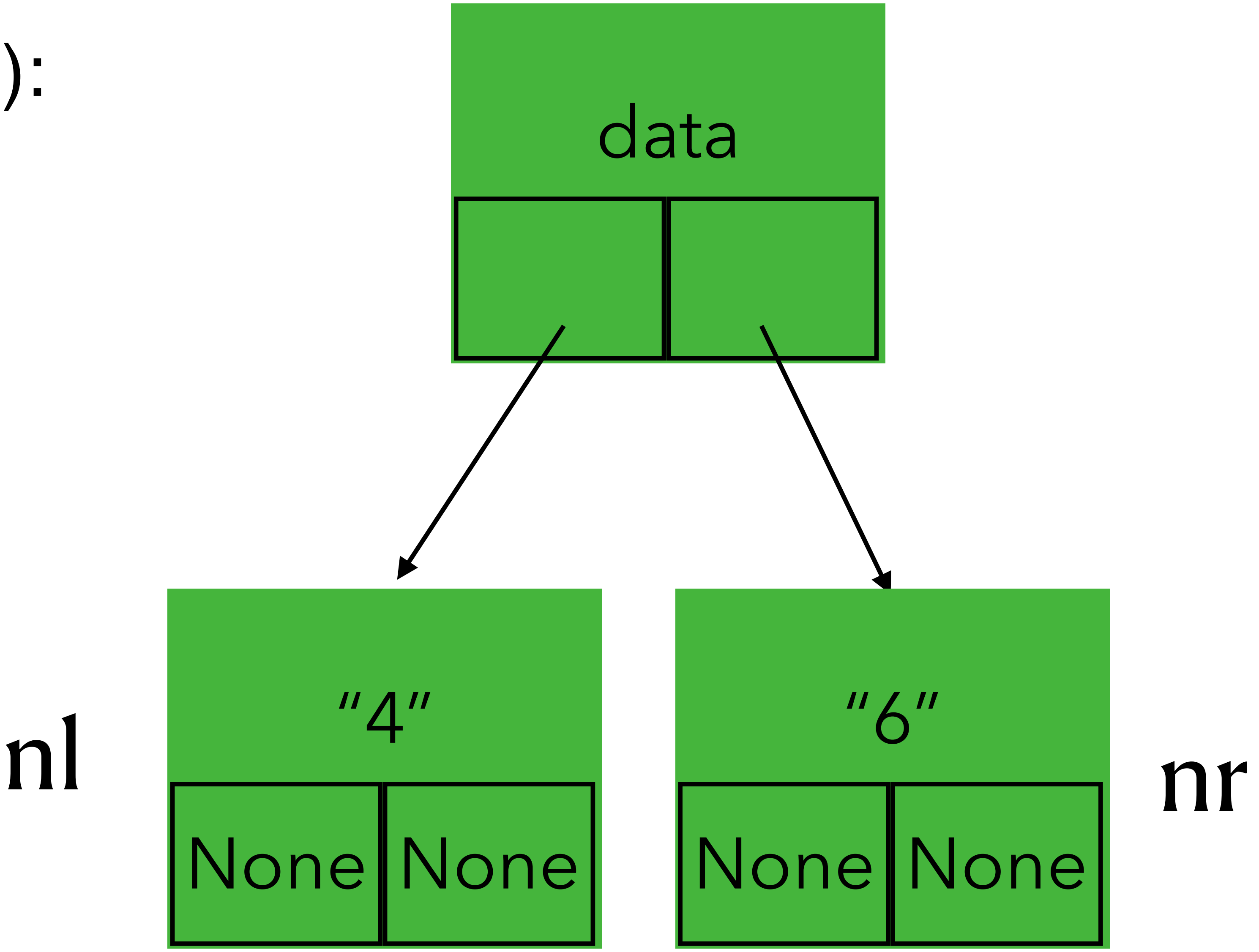
None None

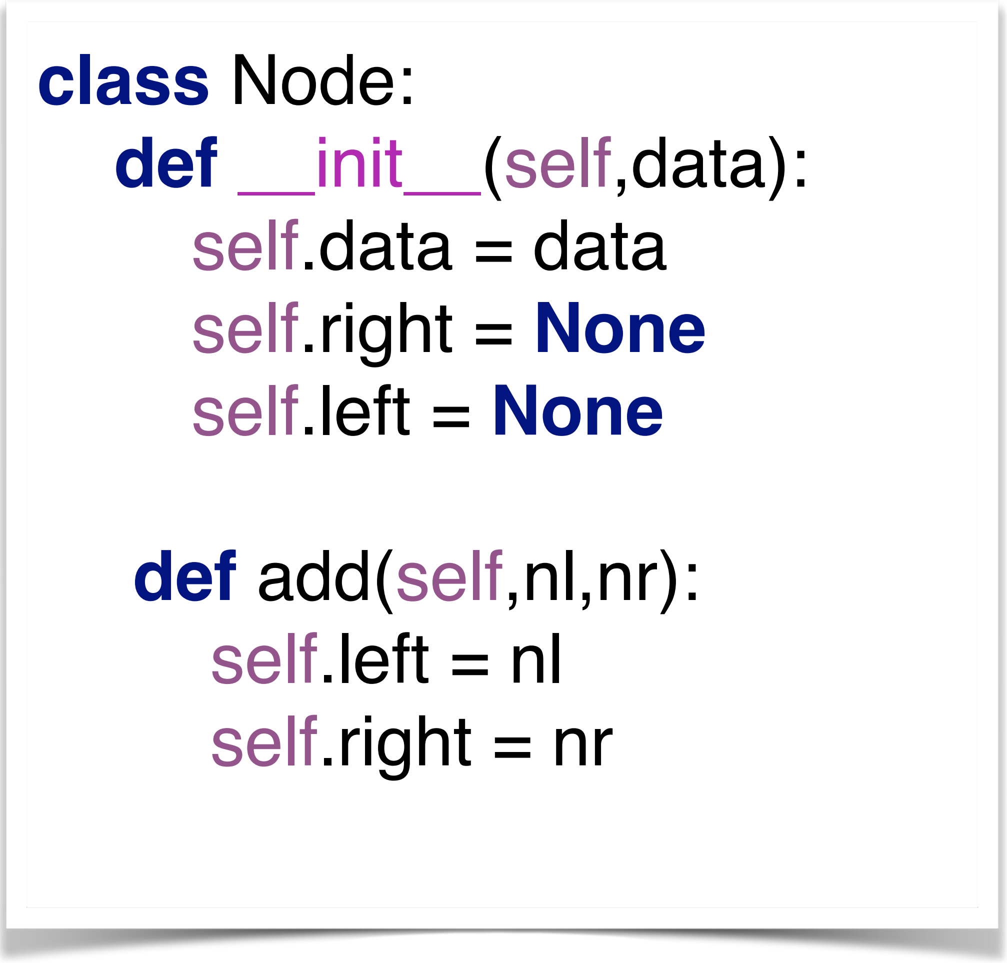
nr “6”

None None

建立二元樹

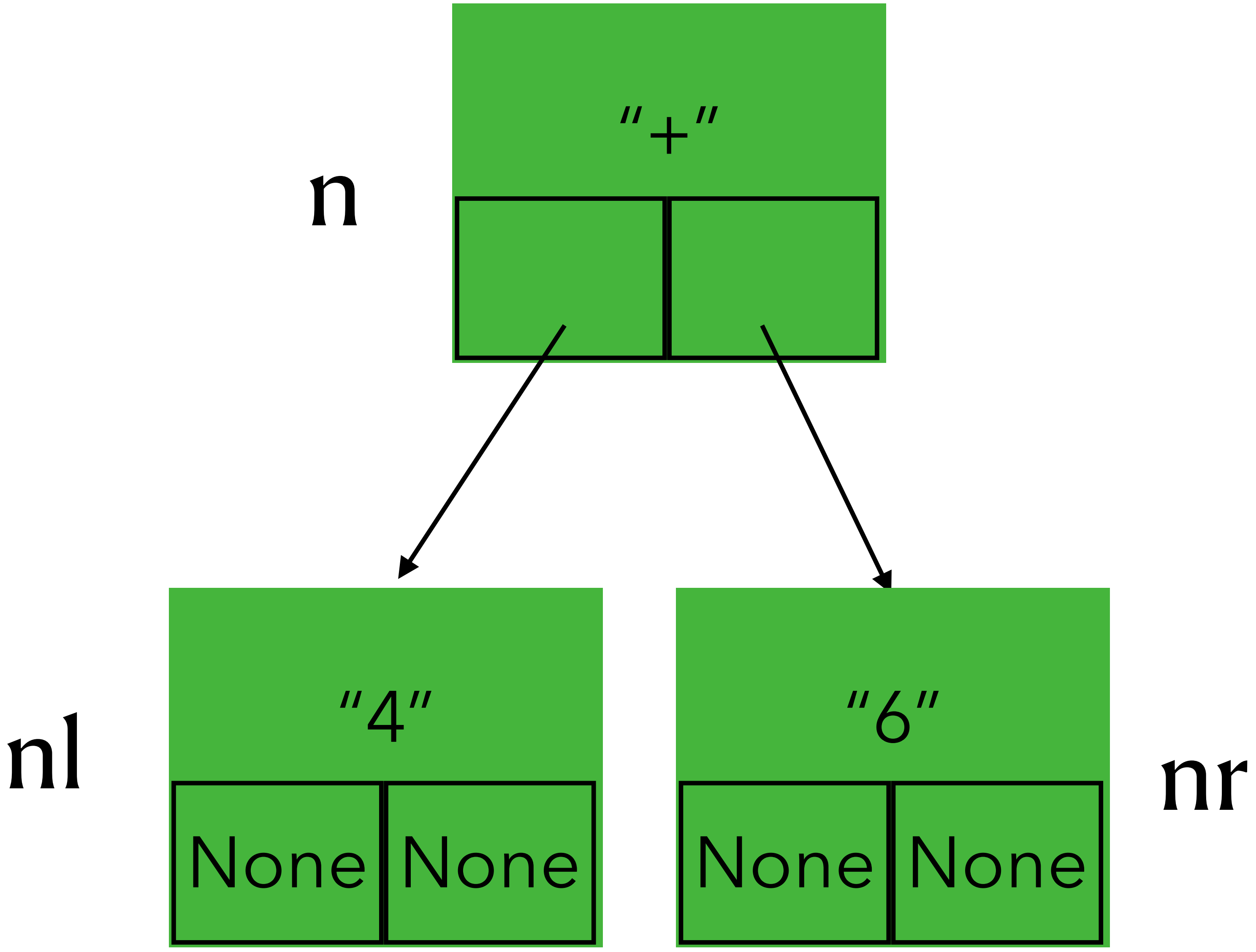
```
class Node:
    def __init__(self,data):
        self.data = data
        self.right = None
        self.left = None

    def add(self,nl,nr):
        self.left = nl
        self.right = nr
```

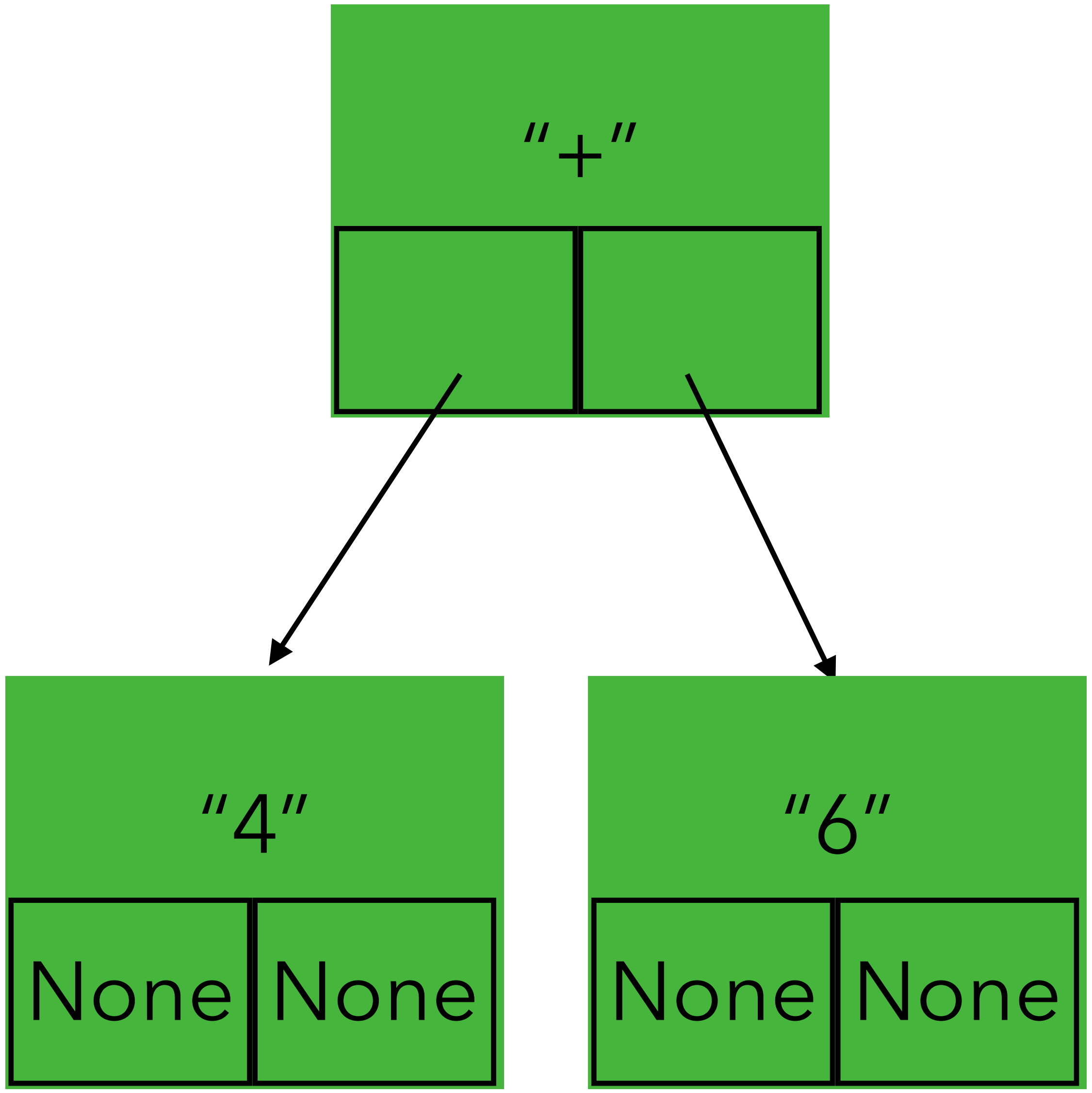


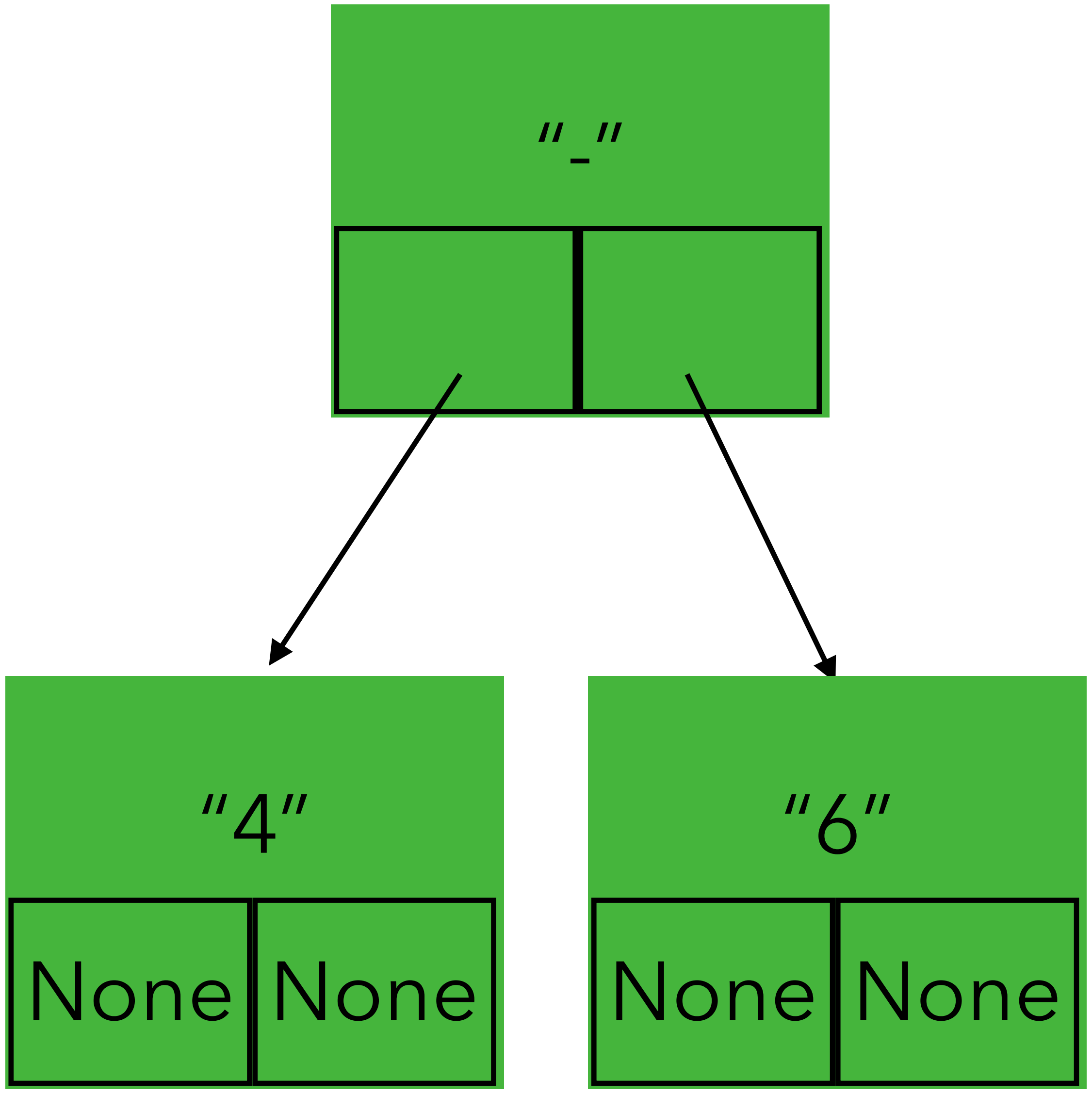


```
class Node:
    def __init__(self,data):
        self.data = data
        self.right = None
        self.left = None

    def add(self,nl,nr):
        self.left = nl
        self.right = nr
```

```
n = Node("+")
nl = Node("4")
nr = Node("6")
n.add(nl,nr)
```

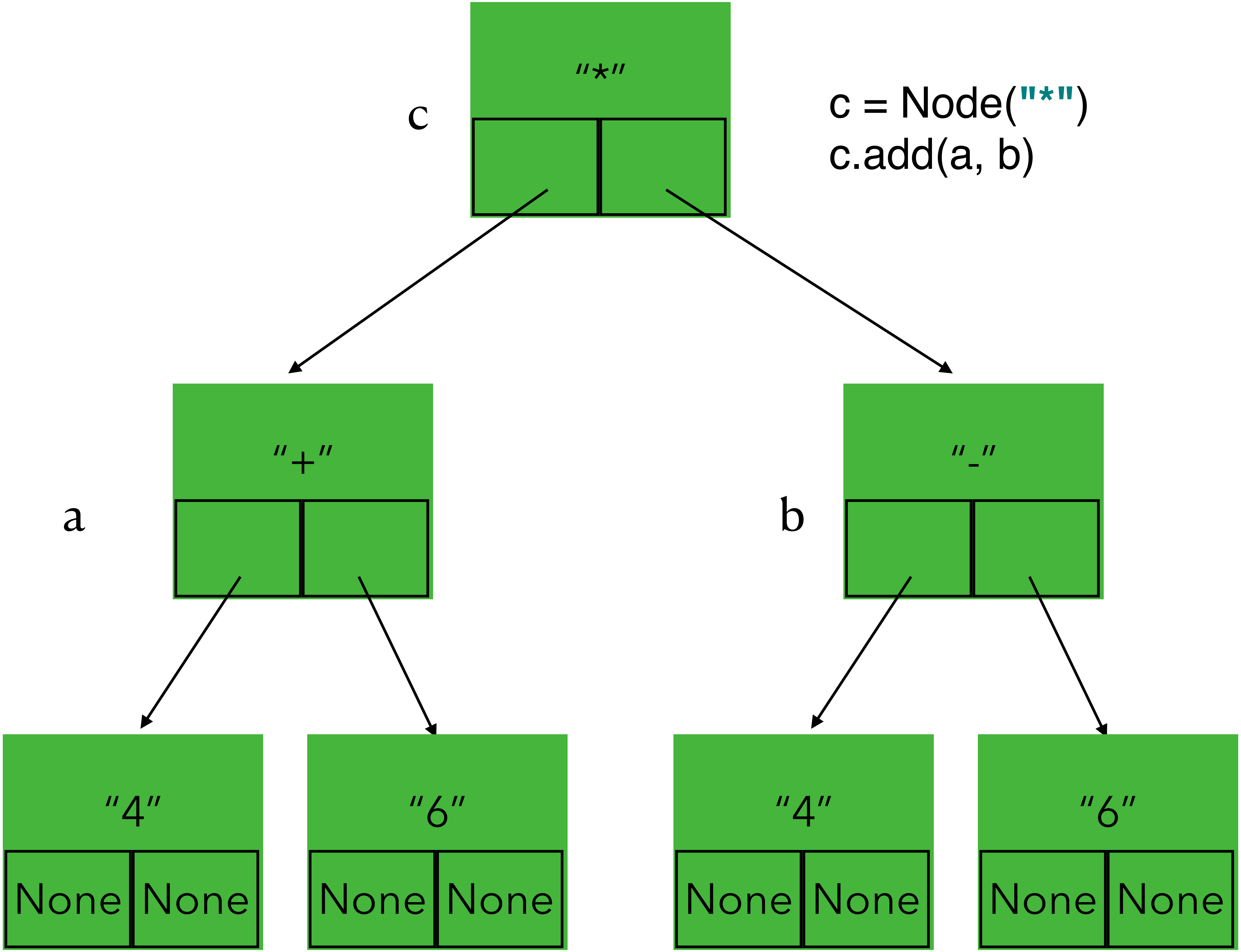




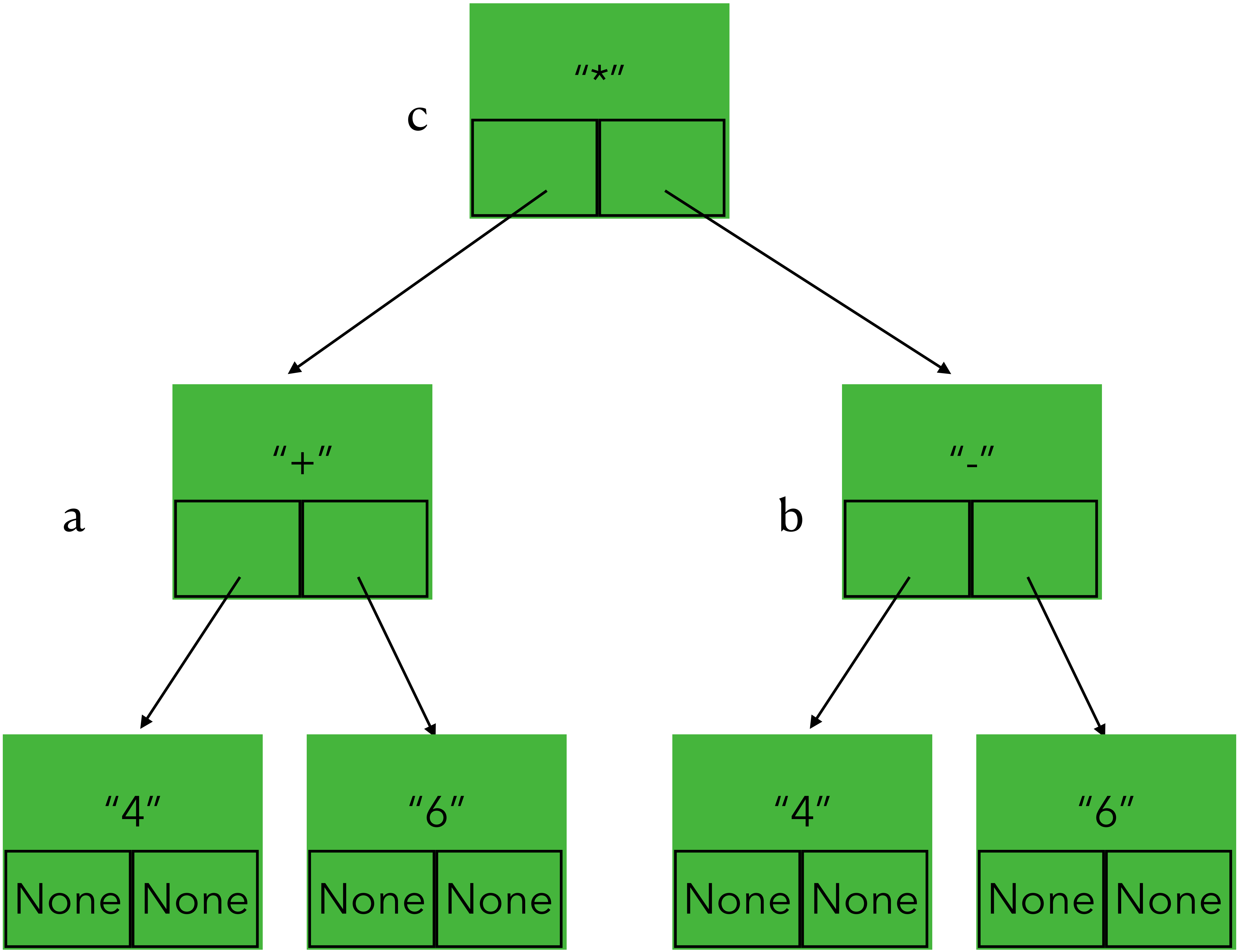


```
a = Node("+")
nl = Node("4")
nr = Node("6")
a.add(nl,nr)
```

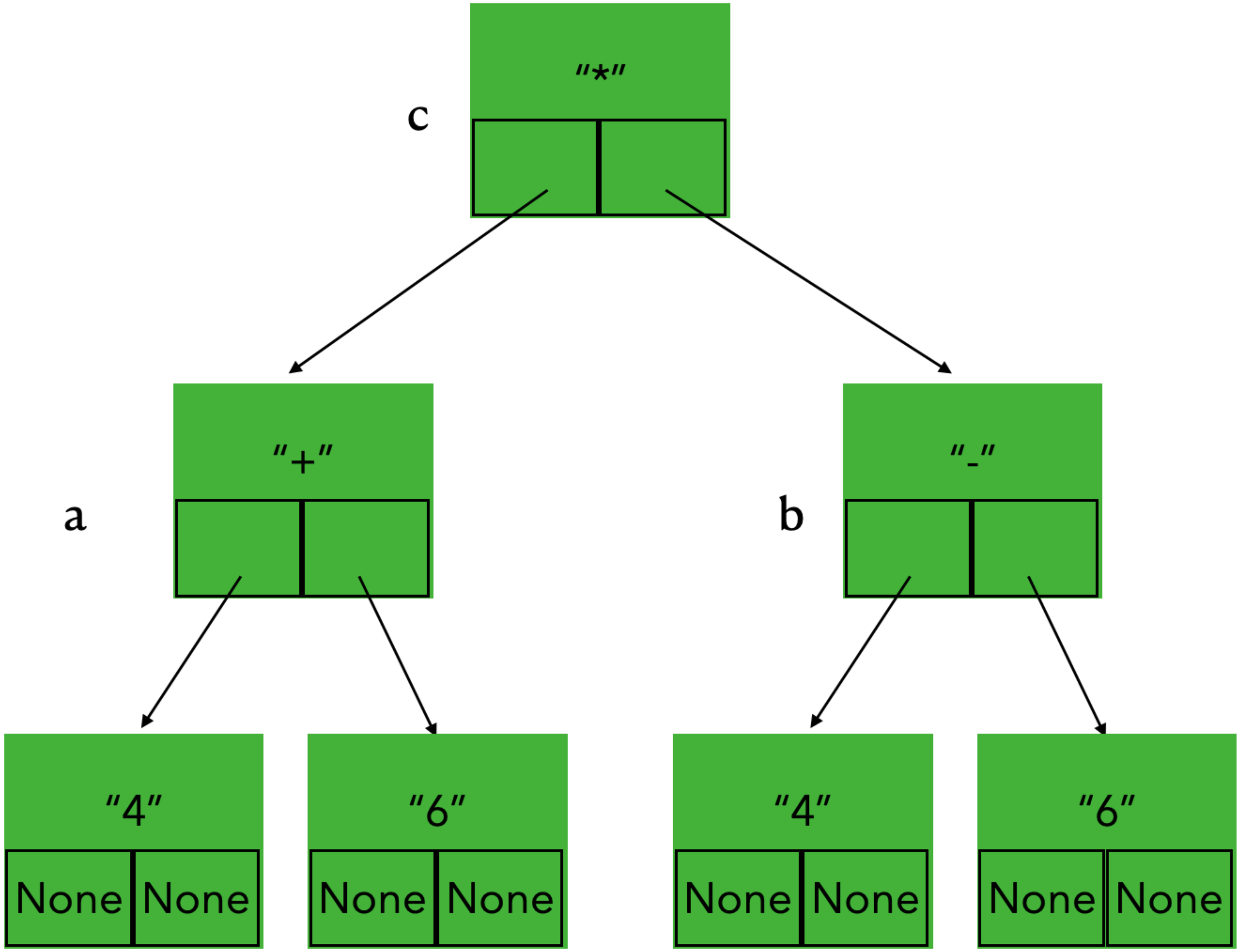
```
b = Node("-")
nl = Node("4")
nr = Node("6")
b.add(nl,nr)
```



```
c = Node("*")
c.add(a, b)
```

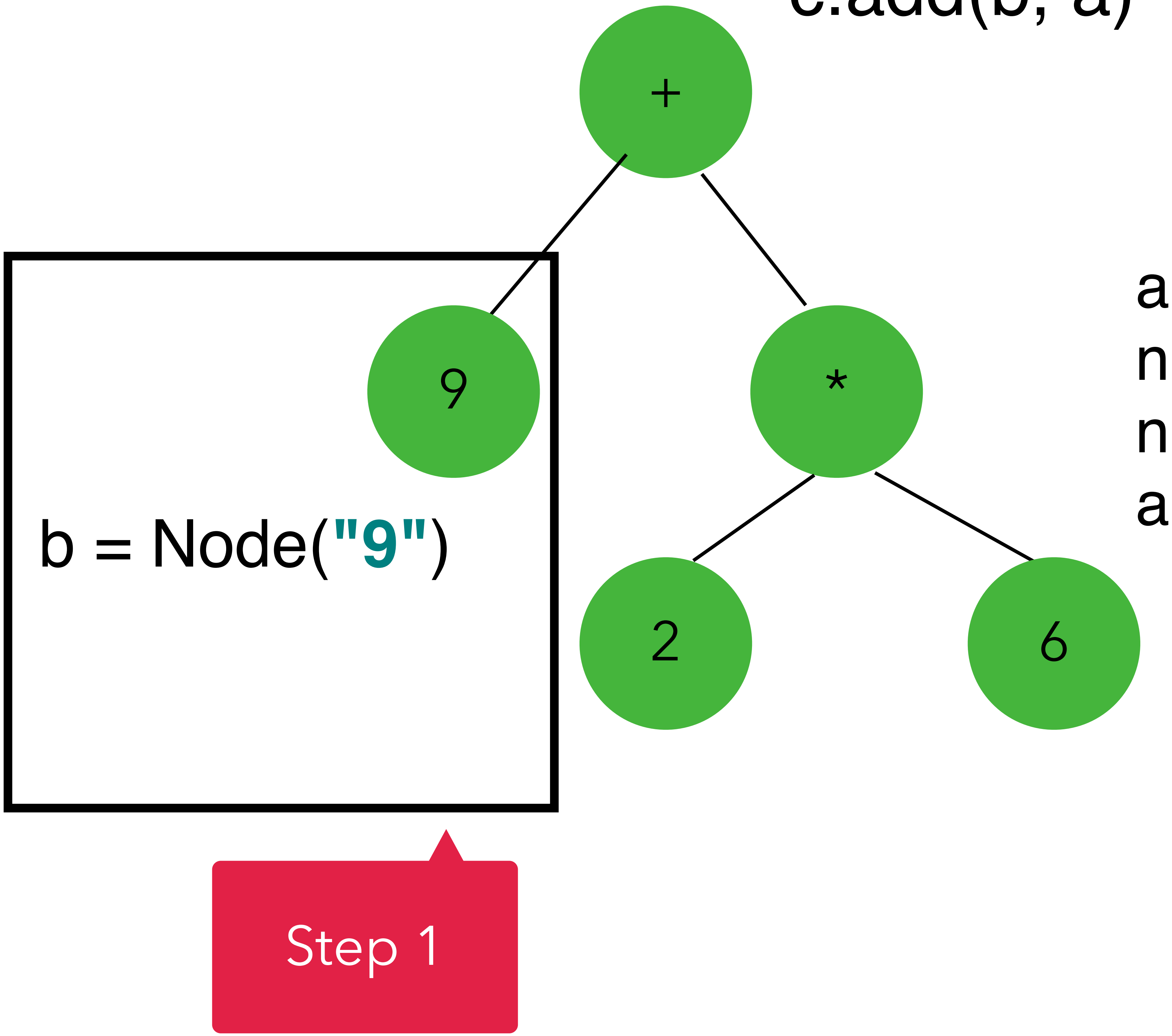


c
"*"
a
"+"
b
"-"
"4"
None
None
"6"
None
None
"4"
None
None
"6"
None
None



```
a = Node("+")
nl = Node("4")
nr = Node("6")
a.add(nl,nr)

b = Node("-")
nl = Node("4")
nr = Node("6")
b.add(nl,nr)

c = Node("*")
c.add(a, b)
```

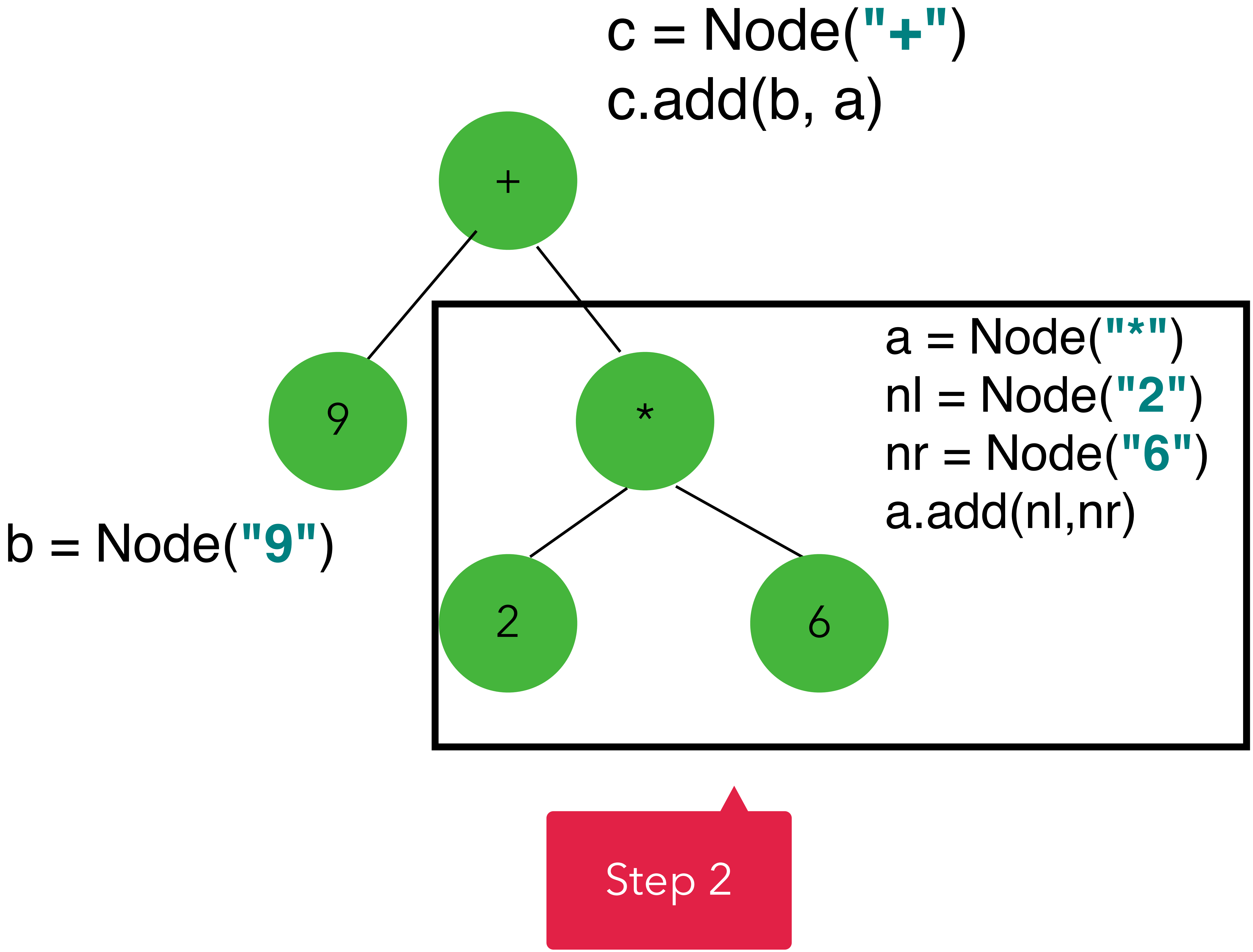
```
c = Node("+")
c.add(b, a)
```



```
b = Node("9")
```

Step 1

```
a = Node("*")
nl = Node("2")
nr = Node("6")
a.add(nl,nr)
```



```
c = Node("+")
c.add(b, a)
```

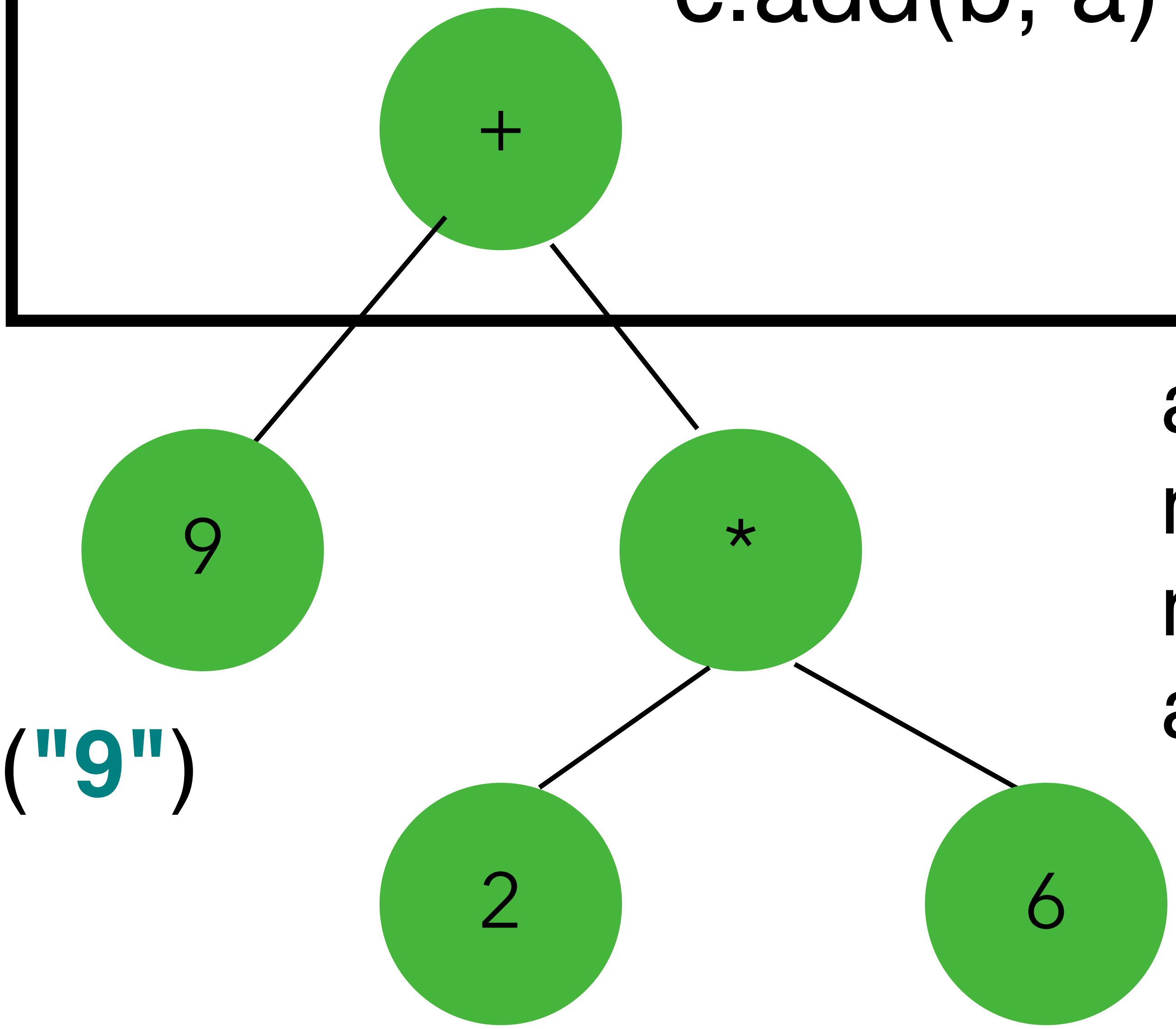
```
b = Node("9")
```

```
a = Node("*")
nl = Node("2")
nr = Node("6")
a.add(nl,nr)
```

Step 2

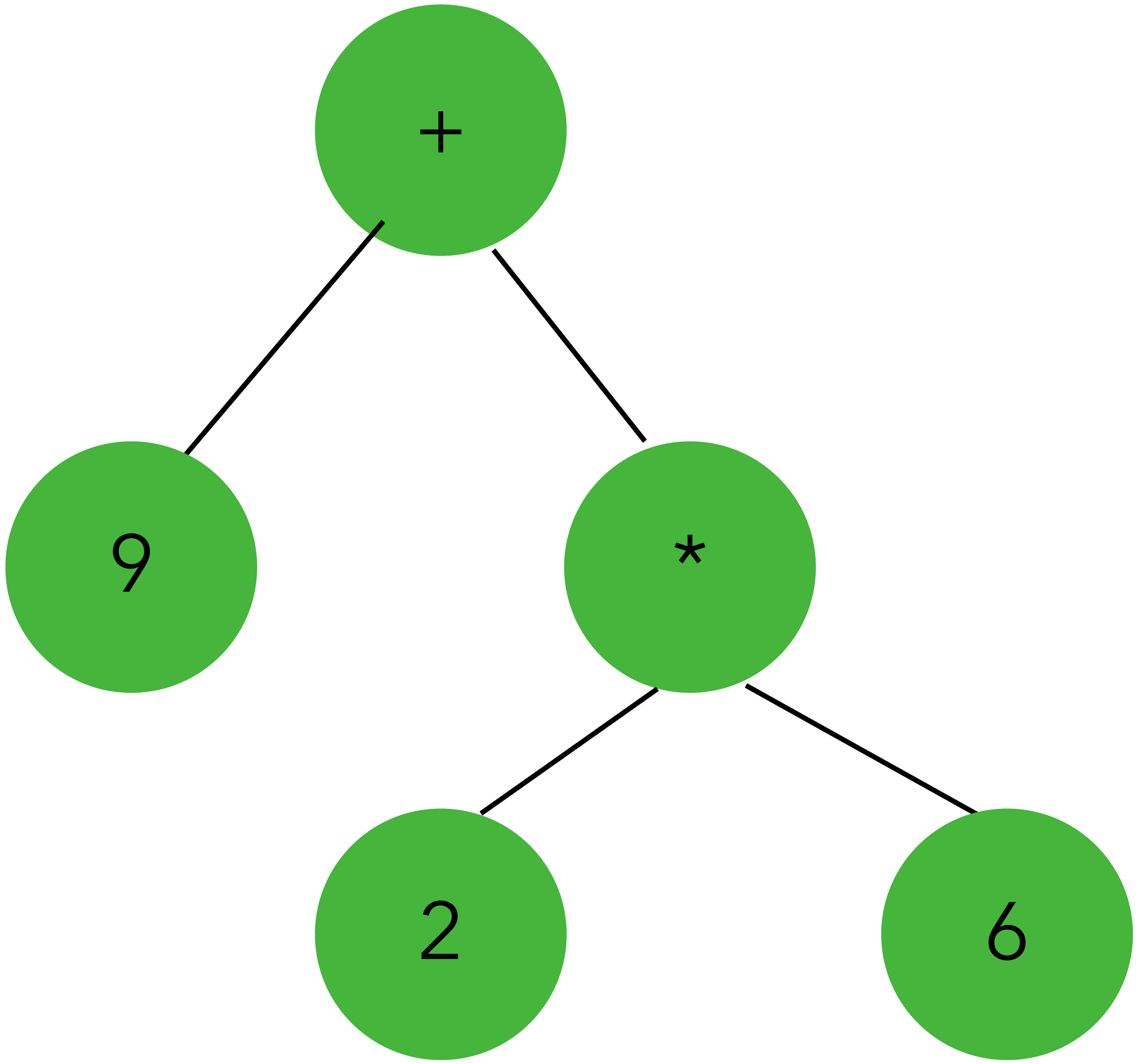
Step 3

```
c = Node("+")
c.add(b, a)
```



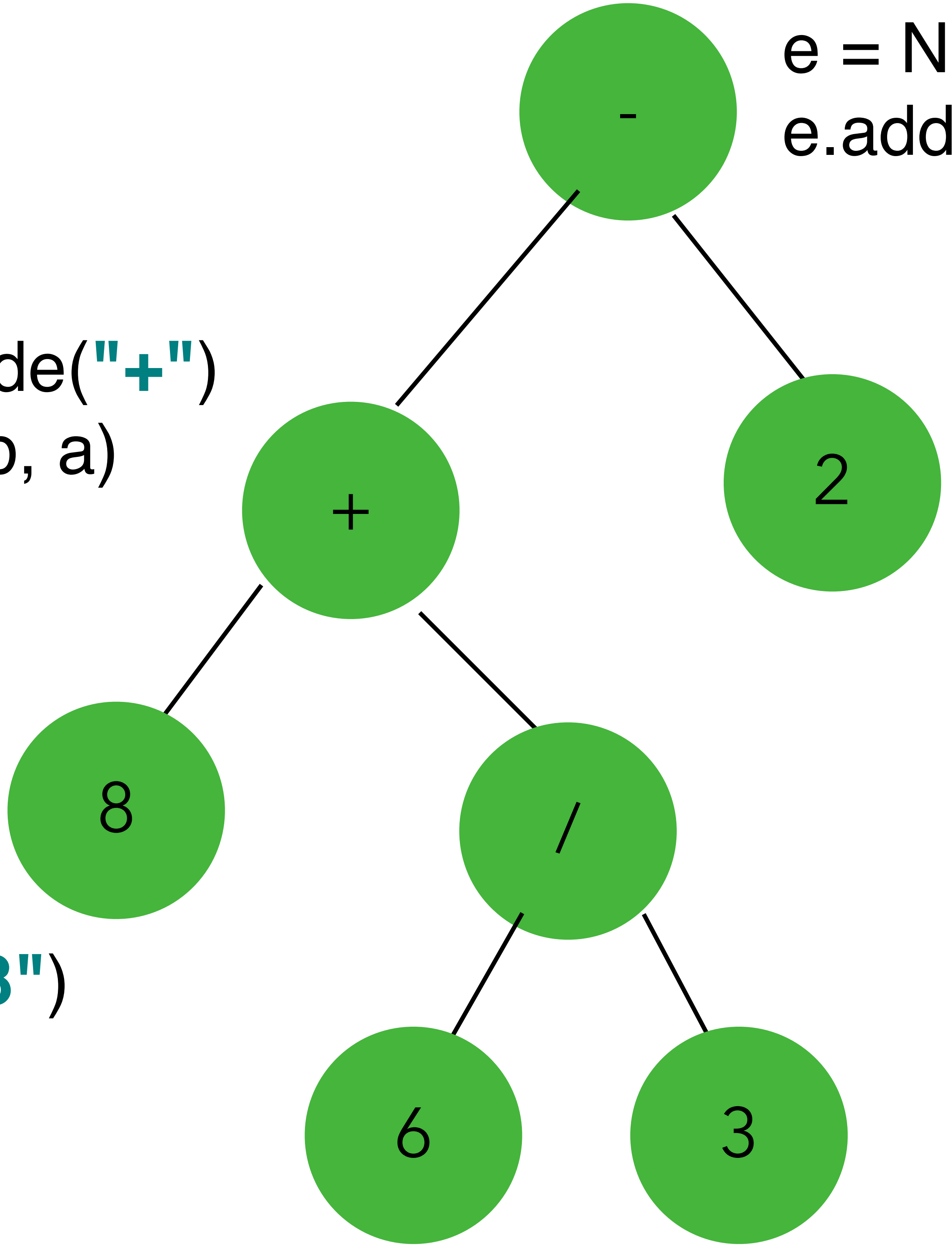
```
a = Node("*")
nl = Node("2")
nr = Node("6")
a.add(nl,nr)
```

```
b = Node("9")
```



```
from node import Node
a = Node("*")
nl = Node("2")
nr = Node("6")
a.add(nl,nr)

b = Node("9")

c = Node("+")
c.add(b, a)
print(c.evalBT())
```

```
21
```



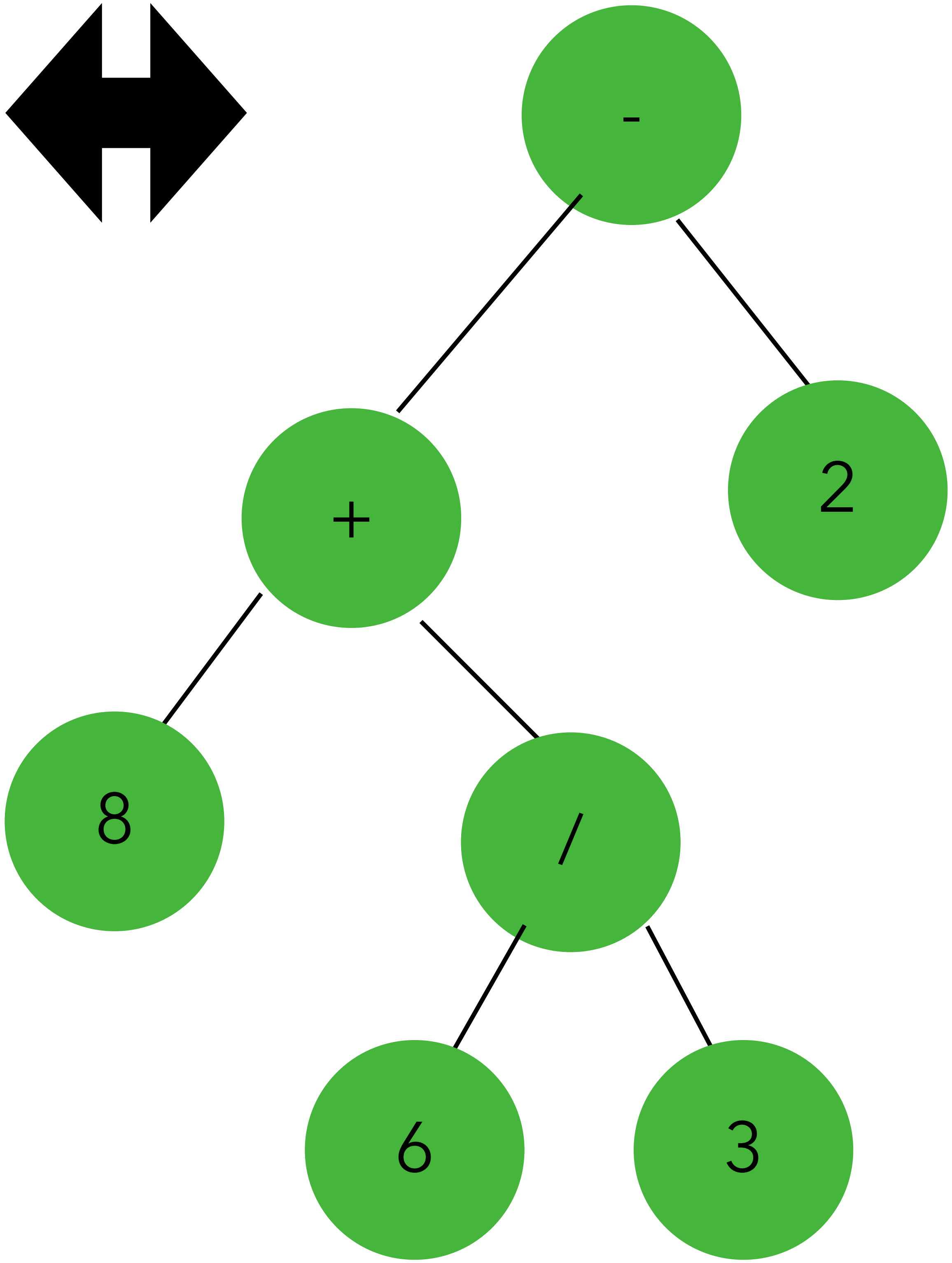
```
e = Node("-")
e.add(c,d)
```

```
c = Node("+")
c.add(b, a)
```

```
d = Node("2")
```

```
b = Node("8")
```

```
a = Node("/")
nl = Node("6")
nr = Node("3")
a.add(nl,nr)
```



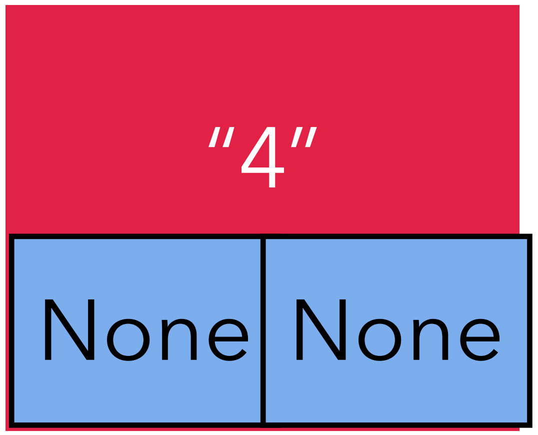
```
from node import Node
a = Node("/")
nl = Node("6")
nr = Node("3")
a.add(nl,nr)

b = Node("8")

c = Node("+")
c.add(b, a)
d = Node(“2”)
e = Node("-")
e.add(c,d)
print(e.evalBT())
```

```
8.0
```

如何以遞迴程式設計 實作方法 evalBT()，回傳二元樹的運算答案

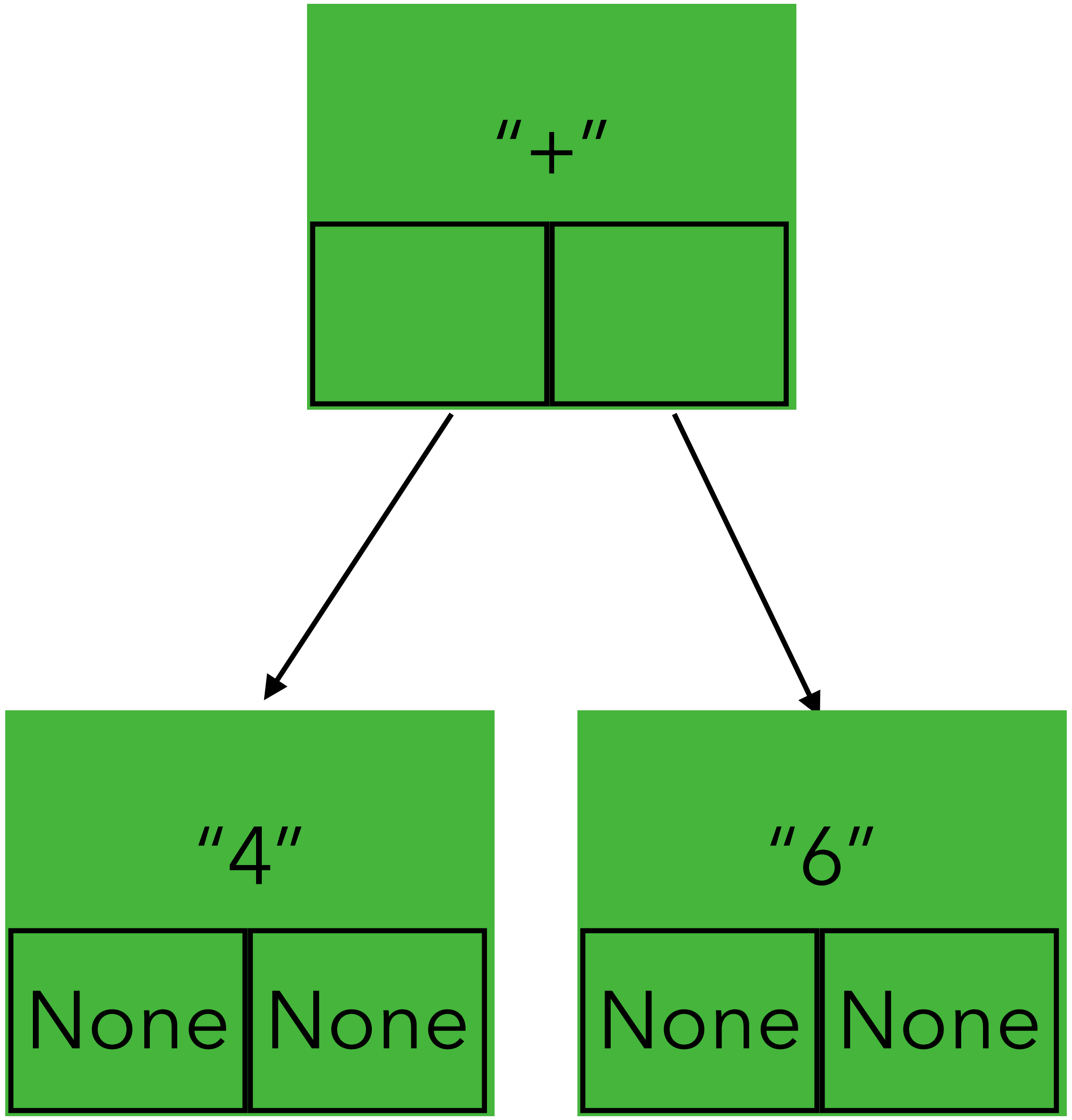
初始問題，直接代入答案

```
def evalBT(self):
    if self.data.isdigit():
        return self.data
```

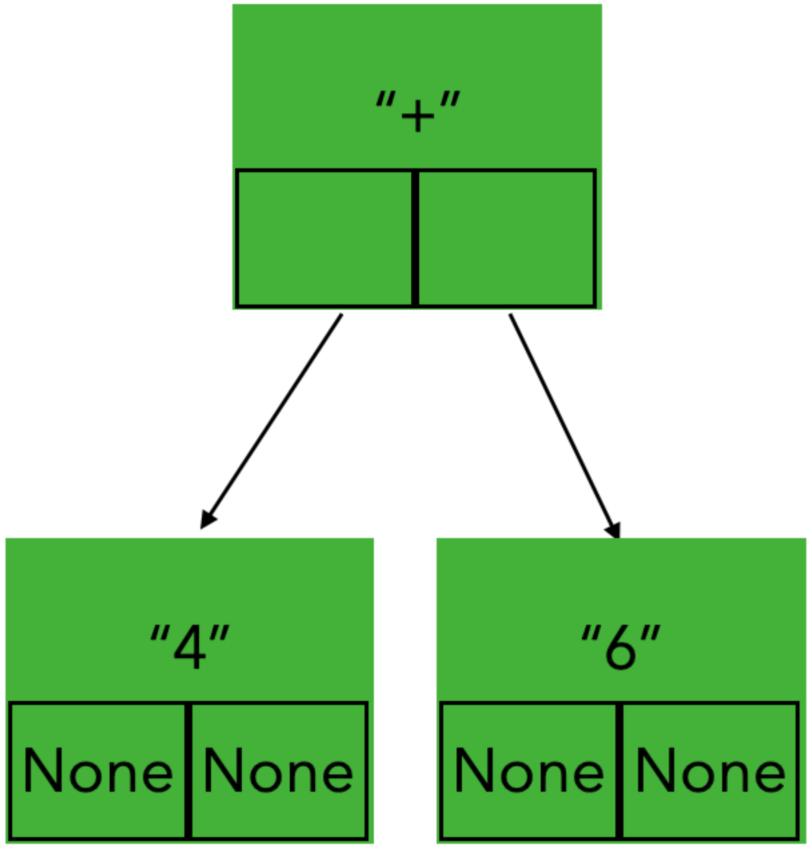


“4”

None None



"+"
"4"
None
None
"6"
None
None



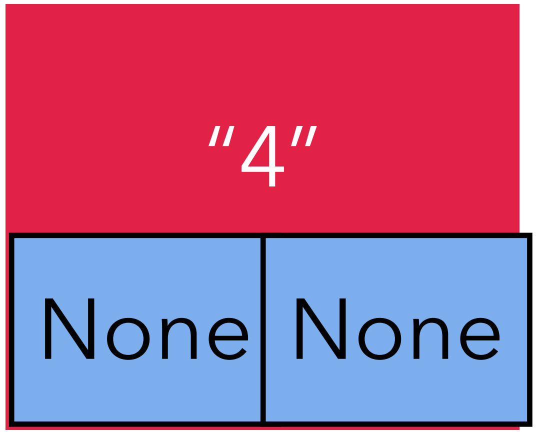
```
def evalBT(self):
    if self.data.isdigit():
        return self.data
    else:
        l = self.left.evalBT()
        r = self.right.evalBT()
        return str(eval(l+self.data+r))
```

不是初始問題，使用前一階問題的答案，合成目前的答案表示式

如何以遞迴程式設計 實作方法 traceBT()，回傳二元樹的中置式運算式

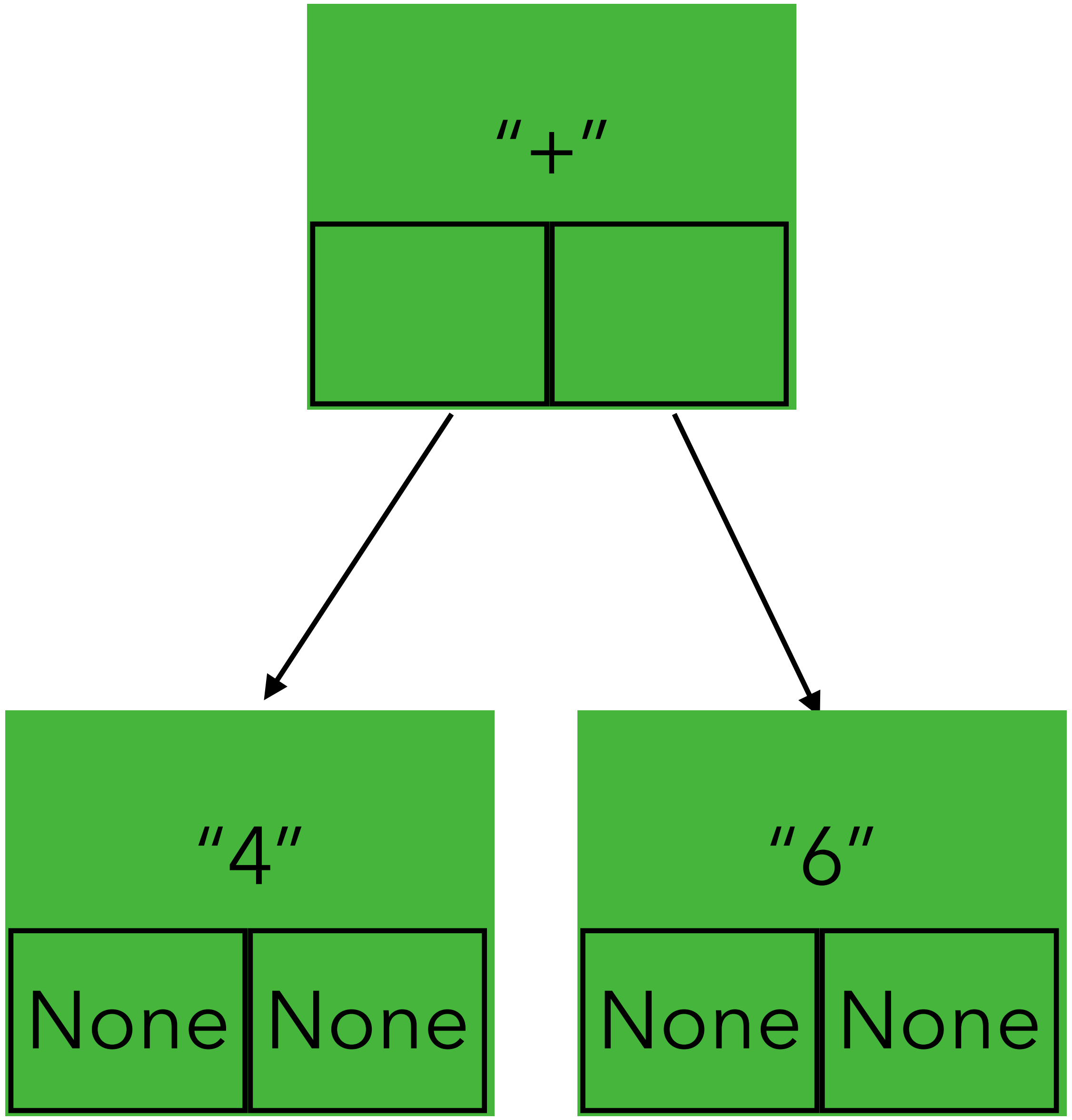
初始問題，直接代入答案

```
def traceBT(self):
    if self.data.isdigit():
        return self.data
```



“4”

None None



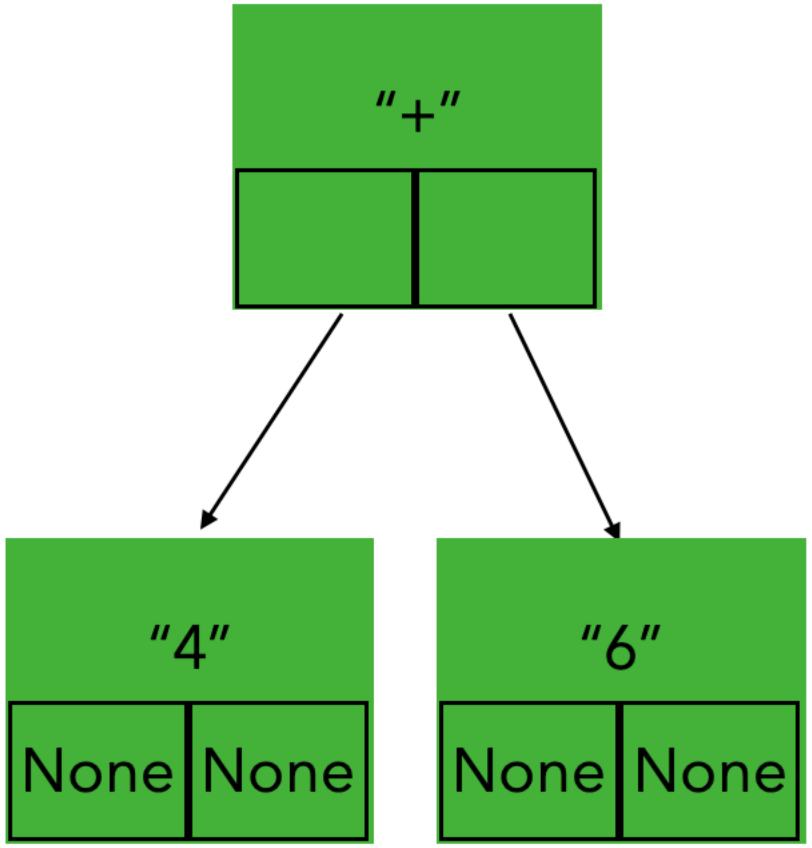
"+"

"4"

None None

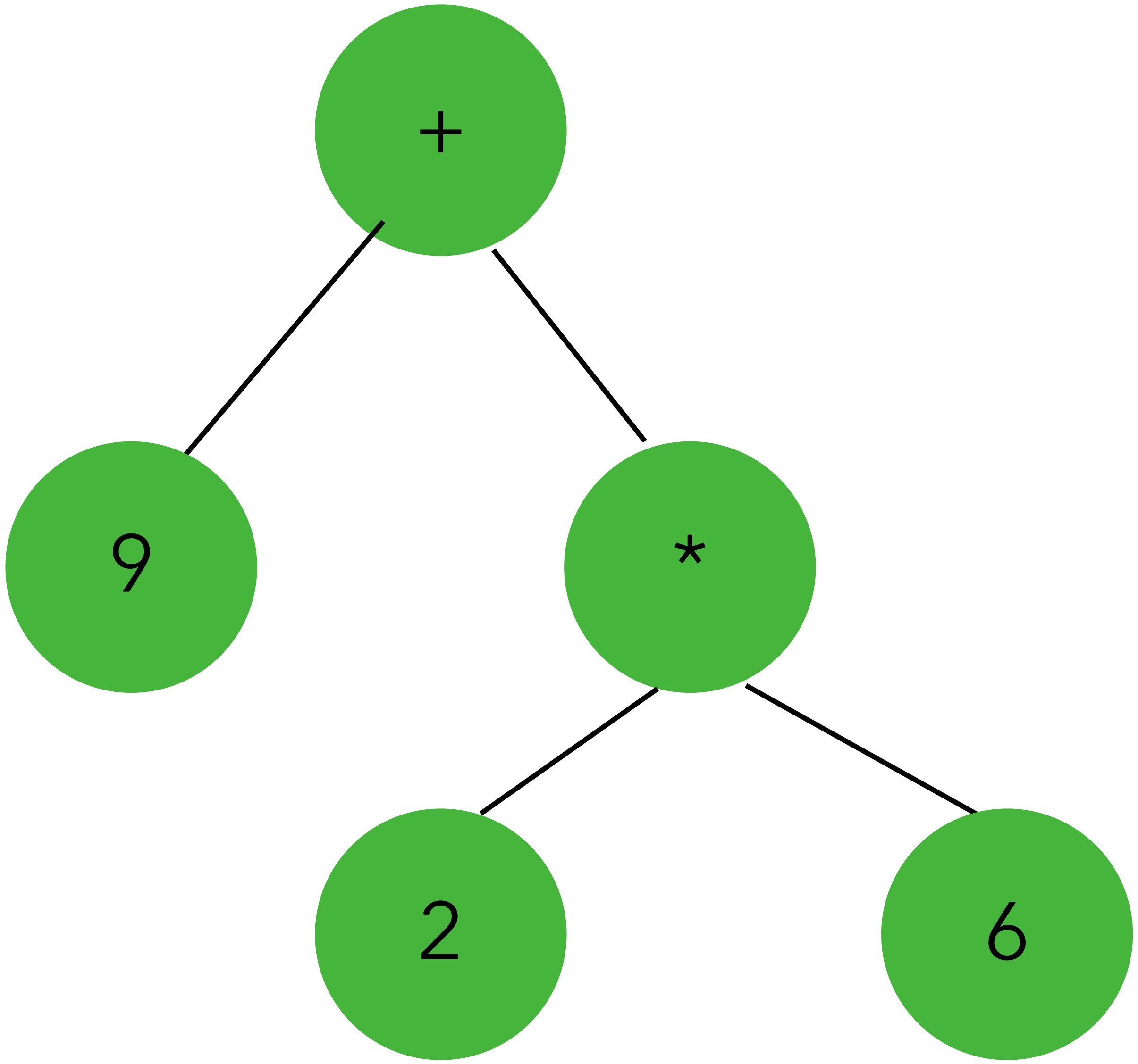
"6"

None None

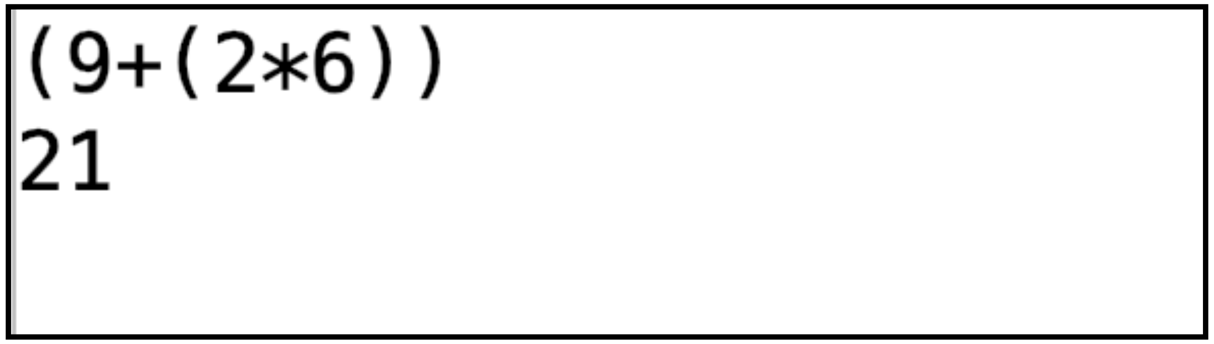


```
def traceBT(self):
    if self.data.isdigit():
        return self.data
    else:
        l = self.left.traceBT()
        r = self.right.traceBT()
        return "("+l+self.data+r+")"
```

不是初始問題，使用前一階問題的答案，合成目前的答案表示式



```
from node import Node
a = Node("*")
nl = Node("2")
nr = Node("6")
a.add(nl,nr)

b = Node("9")

c = Node("+")
c.add(b, a)
print(c.traceBT())
print(c.evalBT())
```



```
(9+(2*6))
21
```