

[Python] __init__(??? ??) & ??? (?????? 25? 1? ??)

17. [] [] [] [] . [] [] [] [] [] [] [] .

```
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []

def tree(li):
    nodes = [Node(i) for i in li]
    for i in range(1, len(li)):
        nodes[(i - 1) // 2].children.append(nodes[i])
    return nodes[0]

def calc(node, level=0):
    if node is None:
        return 0
    return (node.value if level % 2 == 1 else 0) + sum(calc(n, level + 1) for n in
node.children)

li = [3, 5, 8, 12, 15, 18, 21]

root = tree(li)

print(calc(root))
```

- __init__: [] [] [] [] [] . Node [] [] [] []
- children: [] [] [] []
- tree(): [] [] [] [] []
- calc(): [] [] [] [] [] []

13

이 i 1인 경우 ?

li = [3, 5, 8, 12, 15, 18, 21] 이고 i = 0일 때 nodes[(0 - 1) // 2] = nodes[-1]은 배열의 마지막 원소인 21을 반환한다. nodes[0]은 배열의 첫 번째 원소인 3을 반환한다. (root)는 배열의 첫 번째 원소인 3을 반환한다. , nodes[i]는 배열의 i 번째 원소를 반환한다.

18, 21은 2의 거듭제곱이므로, 이들을 2의 거듭제곱으로 올린 후 2를 빼면 된다.

“이 코드는 배열을 사용하여 이진 트리를 표현한다.”

- 이진 트리를 배열로 표현할 때, 노드의 인덱스와 레벨을 계산하는 방법을 알아야 한다.
- 이진 트리의 루트 노드는 인덱스 0에 위치한다.

노드 인덱스	노드 레벨
nodes(index)	노드 인덱스 (0부터 시작)
nodes(level)	노드 레벨 (0부터 시작)

- index 3 (노드 12)의 부모 노드는 index 1이다. $1 \rightarrow 3$
- index 4 (노드 15)의 부모 노드는 index 1이다. $1 \rightarrow 4$
- index 5 (노드 18)의 부모 노드는 index 2이다. $2 \rightarrow 5$
- index 6 (노드 21)의 부모 노드는 index 2이다. $2 \rightarrow 6$

if node is None: 이 조건문을 추가한다.

- node가 None이면, "노드가 존재하지 않음"이라는 메시지를 출력하고, 해당 노드의 부모 노드를 반환한다.
- if node is None: → "노드가 존재하지 않음"이라는 메시지를 출력하고, 해당 노드의 부모 노드를 반환한다.
- 이진 트리를 배열로 표현할 때, 노드의 인덱스와 레벨을 계산하는 방법을 알아야 한다.

node가 None이면, "노드가 존재하지 않음"이라는 메시지를 출력하고, 해당 노드의 부모 노드를 반환한다.

```
3
 / \
5   8
 / \
12  15
```

12 15 18 21 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995

if node is None:
 return 0 # 0을 반환함

1. ??? Node

```
class Node:  
    def __init__(self, value):  
        self.value = value # 0을 반환함  
        self.children = [] # 0을 반환함 (0을 반환함)
```

- Node: 0을 반환함 (0을 반환함)
- value: 0을 반환함 (0을 반환함)
- children: 0을 반환함 (0을 반환함)

[0]

```
n = Node(3)  
print(n.value) # 3  
print(n.children) # []
```

2. ??? tree()

```
def tree(li):  
    nodes = [Node(i) for i in li]  
    for i in range(1, len(li)):  
        nodes[(i - 1) // 2].children.append(nodes[i])  
    return nodes[0]
```

- li: 0을 반환함 (0을 반환함)
- nodes: 0을 반환함 (0을 반환함)
- nodes[(i - 1) // 2]: 0을 반환함 (0을 반환함)



?: li = [3, 5, 8, 12, 15, 18, 21]

index	:	value	
0	:	3	→ root
1	:	5	→ 3의 왼쪽 자식
2	:	8	→ 3의 오른쪽 자식
3	:	12	→ 5의 왼쪽 자식
4	:	15	→ 5의 오른쪽 자식
5	:	18	→ 8의 왼쪽 자식
6	:	21	→ 8의 오른쪽 자식

3. ?? ?? calc()

```
def calc(node, level=0):
    if node is None:
        return 0
    return (node.value if level % 2 == 1 else 0) + sum(calc(n, level + 1) for n in
node.children)
```

- node가 None이면 0을 반환한다.
- **level**이 홀수 (1, 3, ...)이면 node의 value를 반환하고, 짝수이면 0을 반환한다.

-       1 value   $5 + 8 = 13$

