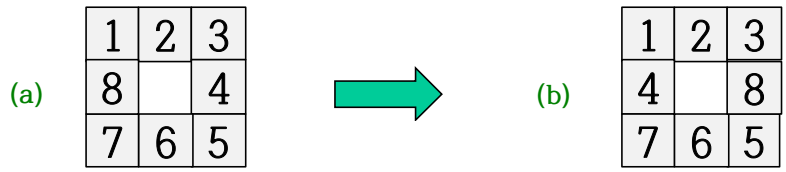
이번 예습자료에서는 **게임 트리**가 무엇이며 어떻게 만드는지를 알아보겠습니다. 이어지는 수업에서는 이러한 게임 트리를 활용해 승리 전략을 도출하는 방법을 배울 것입니다.

<게임 트리의 정의>

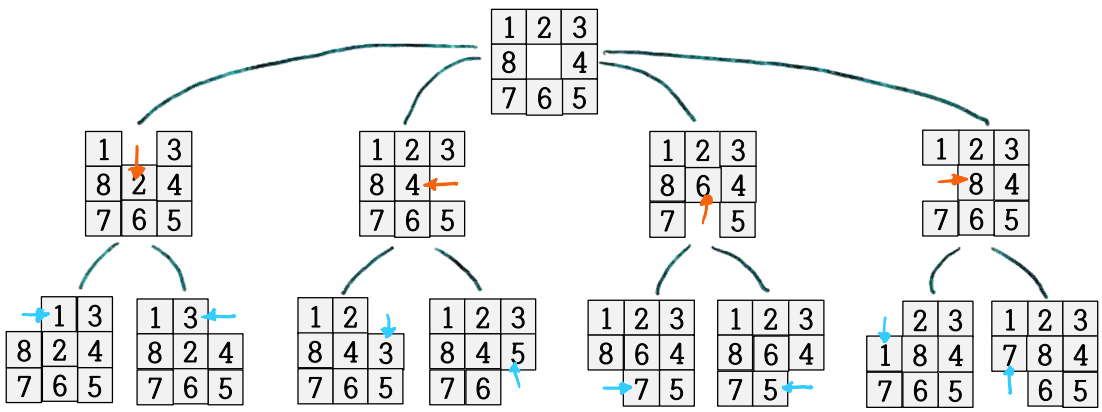
게임 트리는 게임의 참여자(앞으로 player라 하겠음)들이 **게임을 진행하는 과정에서 도달할 수 있는 상태**를 보여주는 **가능성 트리(possibility tree)** 입니다. 특히 게임 트리의 각 정점(node)과 간선(edge)은 다음을 나타냅니다.

- **Node**: 게임 진행 과정에서 거쳐 갈 수 있는 **상태(state)**를 나타냅니다.
  - **Root** node: 게임의 **초기 상태**를 나타냅니다. 또는 게임 트리를 게임 중반부에서부터의 가능성을 표현하기 위해 사용할 수도 있는데 이럴 때 root는 게임 중반부의 상태(혹은 현재 상태)를 나타냅니다.
  - **Leaf** node: 게임의 **최종 상태**(승리, 패배, 비김, 성공, 실패 등)를 나타냅니다.
- **간선**: 게임의 **상태를 변화시키는 행위**(action, activity, move)를 나타냅니다.

그림 1과 같이 숫자 타일을 움직이는 게임을 예로 들어 보겠습니다. 3×3 grid에 8개의 숫자 타일이 있고, 한 번에 하나의 타일을 위, 아래, 왼쪽 또는 오른쪽의 빈 곳으로 움직일 수 있습니다. 이 게임의 목표는 8개의 타일에 대한 임의의 배치가 시작점으로 주어졌을 때(그림 1(a)) 주어진 또 다른 배치가 될 때까지 (그림 1(b)) 타일을 움직이는 것입니다. 특히 최종 배치에 도달할 수 있는 최소 횟수의 이동을 찾는다면 더 좋을 것입니다.



<그림 1. 초기 상태와 최종 상태의 예>



<그림 2. 그림 1의 8-타일 게임에 대한 트리를 depth≤2 까지 그린 예>

그림 2는 이 게임에 대한 트리를 보여줍니다. 그림 1(a)의 시작점을 root로 하여 가능한 모든 이동을 depth≤2까지 그린 것입니다. 단 기존에 이미 거쳐 간 상태로 다시 돌아가지는 않는다고 보았습니다. 예를 들어 root node의 배치에서 시작했으므로 같은 배치로 다시 돌아가는 경우는 그리지 않았습니다. 이러한 조건에 맞게 모든 가능한 경우가 그려졌음을 확인해 보세요. 그림 2 아래로 트리를 계속 그려가다가 그림 1(b)와 같이 우리가 원하는 최종 배치에 도달했다면 root로부터 이 node까지의 경로가 최종 배치에

도달하는 데 필요한 타일의 이동을 나타내게 됩니다.

[Q] 검은색 동전 3개와 흰색 동전 2개가 있다. 두 player A와 B가 번갈아 가며 같은 색의 동전 1~2개를 가져간다. Player A가 먼저 시작해서 동전을 가져갔을 때 다음 상태가 될 수 없는 것은?

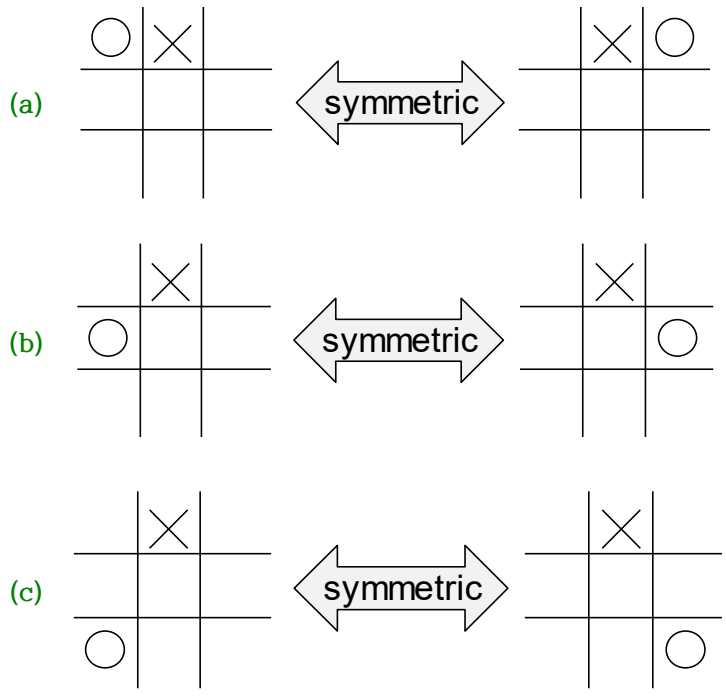
시작 상태:

다음 상태에 대한 보기:

※ 이 자료에서 [Q]로 제시된 문제는 학습 후 풀이하는 온라인 퀴즈에 그대로 나오니 학습하면서 그때그때 문제를 풀어 두세요. 온라인 퀴즈에서 보기의 순서는 바뀔 수 있으니 유의하세요. (예: 보기 1이 보기 2가 되고, 보기 2가 보기 1이 될 수 있음)

<대칭되는 node의 제거>

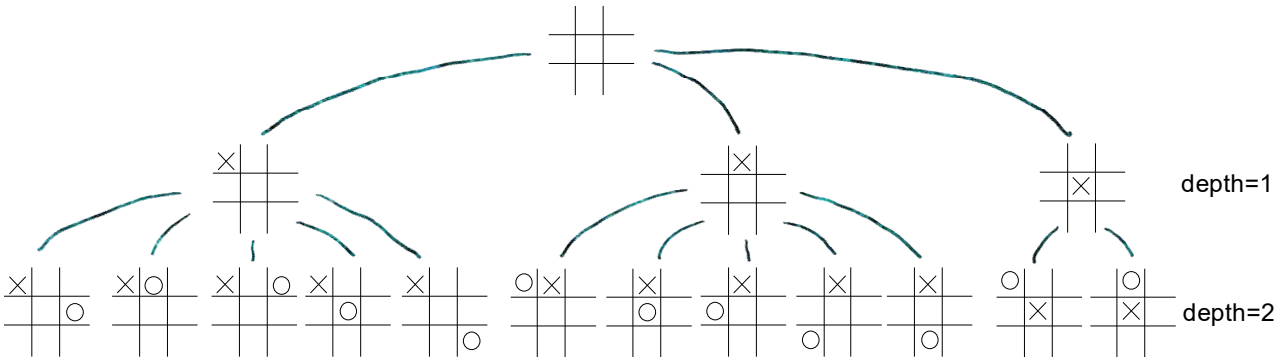
게임 트리를 만들 때 모든 가능성 하나하나를 빠짐없이 포함하면 트리의 크기가 커져서 저장에 많은 메모리가 필요하게 되고 탐색에도 많은 시간이 소요될 수 있습니다. 따라서 정확도를 잃지 않으면서 가능하면 트리를 작게 만들 수 있다면 좋습니다. 이러한 한 가지 방법은 대칭되는 node(symmetric nodes)를 제거하는 것입니다. **두 다른 node를 회전 또는 반전 등을 통해 재배치하였을 때 같은 상태를 나타내는 경우** 이들을 **대칭 node**라 합니다.



<그림 3. 삼목 게임에서 대칭인 상태의 예>

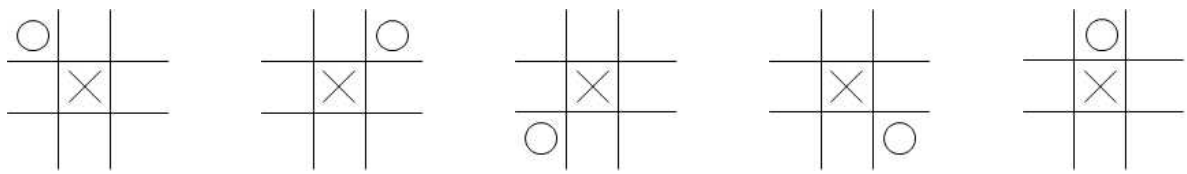
위 그림 3은 삼목 게임(tic-tac-toe game)에서 대칭되는 node의 예 3가지를 보여줍니다. O와 X가 각각 서로 다른 player를 나타냅니다. 대칭되는 node끼리는 ① 서로 반전되는 위치에 있으며 ② **각 player가 승리할 확률이 같고** ③ **이후에 도달할 수 있는 상태도 1대1로 대칭됨**을 확인하세요. 게임 트리를 구성하는 중 이처럼 서로 대칭되는 node n_A와 n_B를 발견하였다면 n_A 아래로만 트리를 그려 속성을 확인한 후 n_B 아래로는 같은 속성이 적용된다고 보면 됩니다.

아래의 그림 4는 삼목 게임에 대한 트리를 보여줍니다. 대칭되는 node를 제거하고 모든 가능한 경우를 depth ≤ 2까지 그린 것입니다. Depth = 1과 depth = 2의 각 node를 보면서 대칭되는 node가 제거되었음을 확인하세요. 예를 들어 depth = 1에서 대칭되는 node를 제거하지 않았다면 총 9개의 node가 존재할 것입니다. 대칭되는 node를 제거한 후에는 그림 4와 같이 3개의 node만 남습니다.



<그림 4. 삼목 게임에 대한 트리를 대칭 node 제거하고 depth ≤ 2까지 그린 예>

[Q] 아래 보기는 삼목 게임의 상태를 보여준다. 이 중 나머지 보기와 대칭이 아닌 것은?

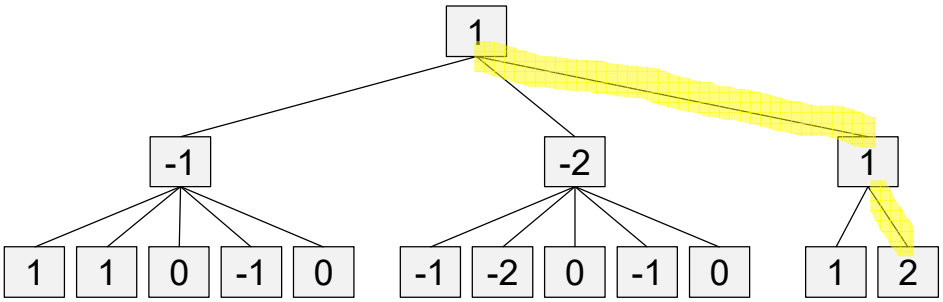


<Evaluation Function>

게임 트리가 준비된 후에는 이를 활용해 승리할 수 있는 전략을 찾아야 합니다. 이를 위해 승리 전략을 찾고자 하는 player가 각 node n에서 얼마나 유리한가를 나타내는 숫자를 부여합니다. 이처럼 숫자를 부여하는 함수를 평가 함수(evaluation function)라 하며, n 값에 따라 숫자가 결정되는 함수이므로 f(n)으로 표기할 수 있습니다. 이러한 숫자가 결정되었다면 player는 현재 상태 $n_{currnet}$의 자식 node 중 f(n)이 가장 큰(즉 player의 승리 가능성이 가장 큰) 상태로 이동하는 action을 취하면 승리(혹은 목표 달성)에 더 가까워질 것입니다.

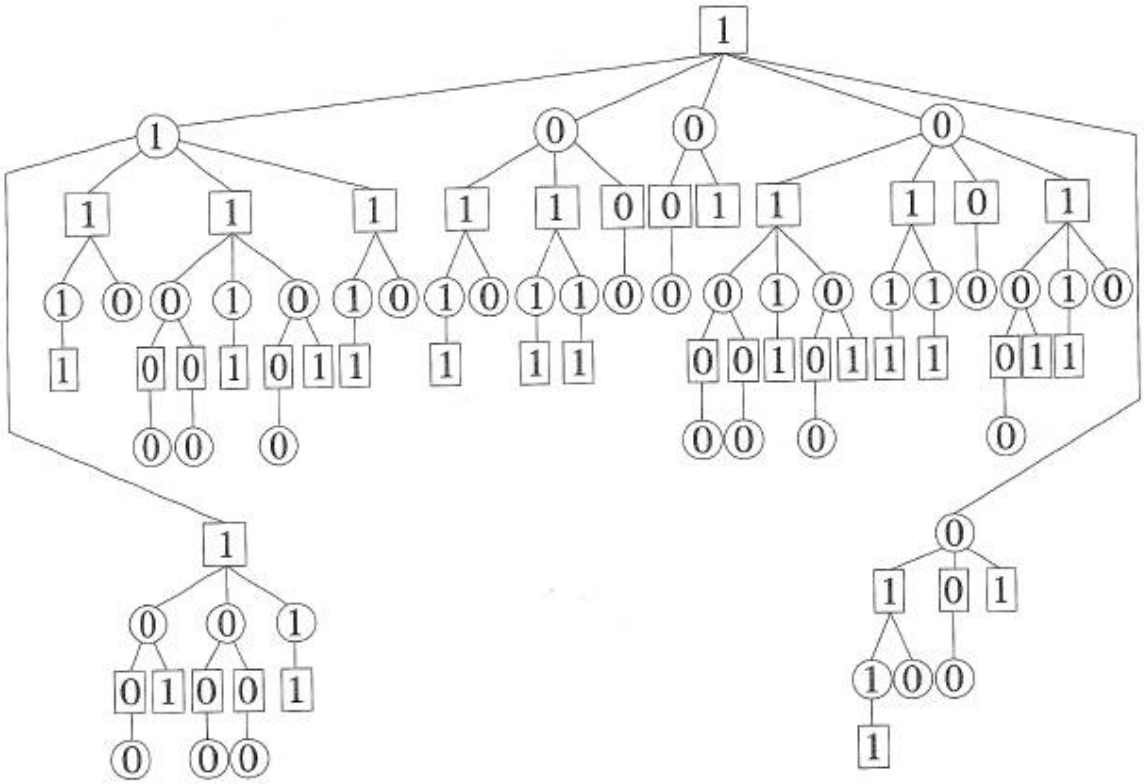
그림 5는 게임 트리의 각 node n에 평가 함수 f(n) 값을 부여한 예입니다. Root node에서 게임을 시작했고 player가 action을 취할 차례라고 하겠습니다. Root의 3개 자식 node 각각의 f(n) 값이 {-1, -2, 1}이므로 이 중 최댓값인 1을 가지는 node 쪽으로 이동(노란색으로 하이라이트 된 경로)하는 것이 이길 확률을 가장 높이는 방법입니다. 이렇게 이동한 후에는 2개 자식 node 각각의 f(n) 값이 {1, 2}이므로 이 중

최댓값이 2를 가지는 node 쪽으로 이동(노란색으로 하이라이트 된 경로)하는 것이 승리 확률을 높일 수 있습니다.



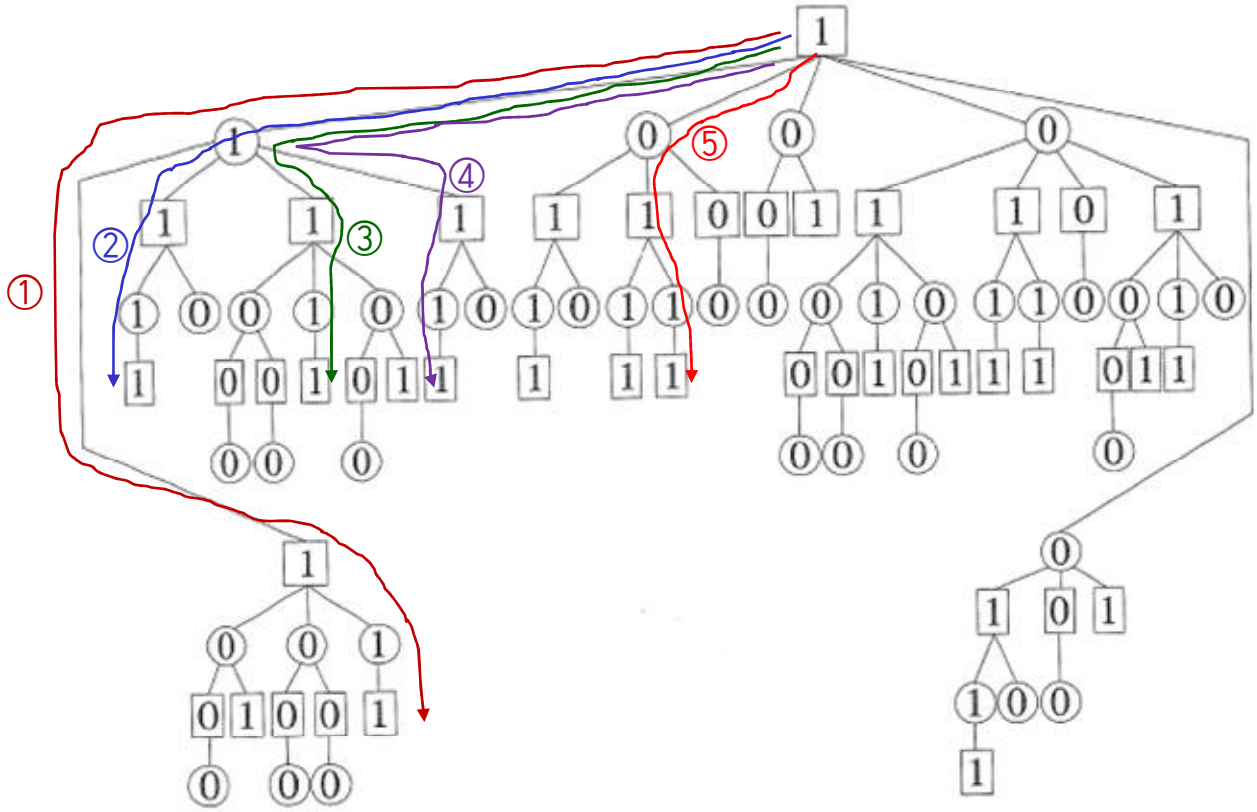
<그림 5. 게임 트리의 각 node에 evaluation function 값을 부여한 예>

적절한 평가 함수 f(*n*)은 게임에 따라 다르며 이를 찾기 위해서는 상당히 많은 실험과 연구가 필요합니다. 이 중 간단하면서 많이 사용되는 평가 함수는 1(승리)과 0(패배)의 두 값을 가지는 함수입니다. 아래 그림 6이 이러한 함수를 부여한 게임 트리의 예를 보여줍니다. f(*root*) = 1이라는 것은 player가 게임에서 이길 방법이 있다는 뜻입니다. 이런 경우 player는 f(*n*) = 1인 node를 따라 게임이 진행되도록 이끌어 가서 최종적으로 f(*n*) = 1인 leaf에 도달하면 게임에서 이길 수 있습니다. 이러한 경로가 하나 이상 있음을 확인해 보세요.



<그림 6. 0-1 Evaluation function을 부여한 게임 트리의 예>

[Q] 아래 그림은 0-1 평가 함수를 게임 트리에 적용한 예를 보여준다. 그림의 경로 ①~⑤ 중 승리하는 방법을 나타내는 경로가 아닌 것은?



<Python 언어에서 이번 시간에 필요한 기능 이해>

지금까지 함수 인자로 **변수**나 **다른 함수의 이름**을 전달하는 경우를 보았습니다. 이 외에도 Python에서는 **클래스 이름**을 함수 인자로 전달할 수 있습니다. 함수를 호출하는 사용자가 함수 내에서 특정 클래스를 사용하도록 지정하고 싶을 때 이처럼 클래스 이름을 함수 인자로 전달합니다. 아래 코드에 이러한 예가 있습니다.

```
00 class TreePlayer:
01     def __init__(self):
02         self.name = "tree"
03
04 class RandomPlayer:
05     def __init__(self):
06         self.name = "random"
07
08 def test(className):
09     t = className()
10     print(t.name)
11
12 if __name__ == "__main__":
13     test(TreePlayer)       # “tree” 출력
14     test(RandomPlayer)     # “random” 출력
```

라인 00~02는 생성자 함수 __init__(self)만을 가지는 클래스 TreePlayer를 정의합니다. 마찬가지로 라인 04~06도 생성자 함수 __init__(self)만을 가지는 클래스 RandomPlayer를 정의합니다. 이들 두 클래스는 멤버 변수 self.name에 각각 다른 값을 ("tree"와 "random")을 할당합니다.

다음으로 라인 08~10은 함수 test(className)을 정의하는데, 클래스 이름(className)을 인자로 받아 라인 09와 같이 이 클래스의 객체 t를 생성한 후, 라인 10과 같이 이 객체의 멤버 변수 name을 출력합니다. 따라서 라인 13과 같이 이 함수의 인자로 TreePlayer 클래스를 전달하면 TreePlayer 객체를 생성해 name에 할당된 "tree"를 출력하고, 라인 14와 같이 RandomPlayer 클래스를 인자로 전달하면 RandomPlayer 객체를 생성해 name에 할당된 "random"을 출력합니다.

다음으로 Python에서 추상 클래스(abstract class)를 생성하고 이를 상속하는 방법을 보겠습니다. 추상 클래스는 자바, C++, C# 등 객체지향 언어에서 배우는 개념입니다. 선언만 하고 구현은 하지 않은 멤버 함수를 추상 함수(abstract method)라 하며, 이러한 추상 함수를 하나라도 포함한 클래스를 추상 클래스라 합니다. 추상 클래스는 아직 구현하지 않은 추상 함수를 포함하므로 객체를 생성할 수 없으며, 이를 상속한 하위 클래스에서 추상 함수를 구현하였다면 하위 클래스의 객체는 생성할 수 있습니다. 이처럼 추상 클래스는 이를 상속한 하위 클래스에게 **반드시 구현해야 하는 함수가 무엇**이며 **이 함수의 형태가 어떠해야 하는지**를 알려주는 역할을 합니다. 아래 코드에 추상 클래스와 이를 상속한 하위 클래스의 예가 있습니다.

```
20 from abc import ABC, abstractmethod
21
22 class Player(ABC): # extends ABC(Abstract Base Class) to become an abstract class
23     @abstractmethod
24     def __init__(self, numBlack, numWhite, player): pass
25
26     @abstractmethod
27     def doMyTurn(self): pass
28
29 class TreePlayer(Player):
30     def __init__(self, numBlack, numWhite, player):
31         self.name = "tree"
32
33     def doMyTurn(self):
34         return 0, 3
35
36 if __name__ == "__main__":
37     p = Player()            # 추상 클래스 객체는 생성할 수 없으므로 오류 발생
38     t = TreePlayer(3, 3, 0) # 추상 클래스에서 상속한 모든 추상 함수를 구현했으므로 객체 생성 가능
39     print(t.doMyTurn())     # TreePlayer 클래스에 구현한 것처럼 0, 3을 반환하여 이를 출력
```

Python에서 추상 클래스를 선언하려면 클래스 ABC(Abstract Base Class)를 상속해야 합니다. 따라서 라인 20과 같이 먼저 ABC 클래스를 import 했습니다. 그리고 라인 22~27과 같이 ABC를 상속한 추상 클래스 Player를 정의했습니다. 이 클래스는 두 개의 멤버 함수 __init__()와 doMyTurn()을 선언만 하고 구현

은 하지 않았습니다. 선언만 하고 구현은 하지 않았다는 것은 함수의 이름과 인자는 기술하였지만 함수 내부에서 어떤 일을 해야 하는지는 기술하지 않고 "pass"만 썼다는 뜻입니다. 따라서 이 두 함수는 추상 함수가 되며, Player 클래스를 상속한 하위 클래스에서 두 함수를 Player에서 기술한 바와 같은 형태로 구현해야 함을 알려주는 역할을 합니다. 추상 함수 앞에는 라인 23, 26과 같이 "@abstractmethod"를 써주어야 함을 유의하세요. 이렇게 추상 함수임을 명시하였다면, 하위 클래스에서 이를 반드시 구현해야 하며, 구현하지 않는다면 하위 클래스 역시 추상 클래스가 되어 객체를 생성할 수 없게 됩니다. 추상 클래스는 아직 구현하지 않은 함수가 있으므로 객체를 생성할 수는 없으며, 따라서 라인 37과 같이 객체를 생성하려고 하면 오류가 발생합니다.

라인 29~34는 앞에서 본 추상 클래스 Player를 상속하고, 상속한 모든 추상 함수를 구현한 클래스 TreePlayer를 보여줍니다. 라인 30~31은 함수 __init__()을 클래스 Player에 선언된 형태 그대로 구현하였으며, 라인 33~34는 함수 doMyTurn()을 역시 클래스 Player에 선언된 형태 그대로 구현하였습니다. 추상 함수를 모두 구현하였으므로 라인 38과 같이 TreePlayer 클래스의 객체를 생성할 수 있으며, 라인 39와 같이 이 객체에 대해 구현된 함수를 호출할 수도 있습니다.

이번에는 Python에서 log 함수를 사용하는 방법을 알아보겠습니다. log 함수는 math 모듈 내에 정의되어 있으므로 math 모듈을 import한 후에 math.log(value, base) 형태로 호출합니다. value에 대한 log를 구하되, log base로 base를 사용하라는 의미입니다. 아래에 사용 예가 있습니다.

```
40 import math
41 print(math.log(math.e))        # 1.0 출력
42 print(math.log(10))            # 2.3025... 출력
43 print(math.log(math.e, 10))   # 0.4342... 출력
44 print(math.log(10, 10))       # 1.0 출력
```

라인 41~42와 같이 base를 생략하면 base로 math.e(오일러 숫자 e = 2.718... 의미)를 사용합니다. 따라서 natural log인 ln(value)를 계산하게 됩니다. 라인 43~44와 같이 base = 10으로 지정하면 이 값을 base로 사용한 log를 계산하게 됩니다. 라인 43은 0.4342...을 출력하는데, math.e가 $10^{0.4342...}$ 라는 뜻이며, 라인 44는 1.0을 출력하는데, $10^{1.0}$ = 10이기 때문입니다.

[Q] 추상 클래스에 대한 설명으로 잘못된 것은?

추상 클래스는 이를 상속한 하위 클래스에게 어떤 함수를 어떤 형태로 구현해야 하는지 알려주는 역할을 함
추상 클래스의 객체를 생성할 수 있음
Python에서 추상 클래스를 만들려면 클래스 ABC를 상속해야 함
추상 함수 앞에는 "@abstractmethod"를 써주어야 추상 함수로 인식됨