What the DP array represents

What does `dp[i][j]` mean?

`dp[i][j]` = "Is it possible to process the first `i` characters of the string and end up with exactly `j` unmatched opening parentheses?"

Let me break this down:

Your Mental Model (Slightly Off)

You said: *"does dp[i][j] signify that current string i at count j can create a valid parenthesis?"*

Almost right, but not quite! `dp[i][j]` doesn't tell us if the string is valid yet - it tells us if a specific **intermediate state** is reachable.

Correct Mental Model

`dp[i][j]` tells us: **"After processing `i` characters, can we have exactly `j` unmatched `(` brackets?"**

Example Walkthrough

Let's trace `s = "(*)"` to see what each `dp[i][j]` means:

```
dp[0][0] = True   # Processed 0 chars, 0 unmatched '(' → Starting state

dp[1][1] = True   # Processed '(', now have 1 unmatched '('
dp[1][0] = False  # Can't have 0 unmatched after seeing '('

dp[2][0] = True   # Processed '(*', * as ')' → 1-1=0 unmatched
dp[2][1] = True   # Processed '(*', * as empty → still 1 unmatched
dp[2][2] = True   # Processed '(*', * as '(' → 1+1=2 unmatched
```

```
dp[3][0] = True   # Processed '(*)', final ')' can match → 0 unmatched ✓
dp[3][1] = False  # After final ')', can't have 1 unmatched
```

The Key Insight

- **Individual** `dp[i][j]`: "Is this intermediate state possible?"
- **Final answer** `dp[n][0]`: "After processing ALL characters, can we have 0 unmatched brackets?" → **This means valid parentheses!**

Why Your Wrong Solution Failed

Issue 1: Wrong Base Case

```
# WRONG
for i in range(n+1):
    dp[i][0] = True# "Any string with 0 unmatched is valid"
```

Problem: This assumes any prefix can magically have 0 unmatched brackets, which isn't true.

Example: For string `"((("`, you can't have 0 unmatched brackets after processing any characters!

Issue 2: Wrong Return

```
return dp[n][n]# WRONG - asking "can we have n unmatched brackets?"
return dp[n][0]# CORRECT - asking "can we have 0 unmatched brackets?"
```

Visual Understanding

Think of it like a **state machine**:

```
Start: 0 unmatched brackets
  ↓
Process each character:
- '(' → increase count
- ')' → decrease count
- '*' → increase/decrease/same count
  ↓
End: Want 0 unmatched brackets for valid string
```

Each `dp[i][j]` tracks: **"Is count=j reachable after i steps?"**

The DP Transition Logic

```
if s[idx-1] == '(':
    dp[idx][count] = dp[idx-1][count-1]# "Add 1 unmatched '('"

elif s[idx-1] == ')':
    dp[idx][count] = dp[idx-1][count+1]# "Remove 1 unmatched '('"

elif s[idx-1] == '*':
# Can transition from count-1, count, or count+1
    dp[idx][count] = dp[idx-1][count-1] or dp[idx-1][count] or dp[idx-1][count+1]
```

The final `dp[n][0]` tells us: **"After all characters, is it possible to have perfect balance?"**