House Robber

Difficulty	Easy
Category	1-DP
Question	https://leetcode.com/problems/house-robber/
Solution	https://youtu.be/73r3KWiEvyk
Status	Done

Question

> You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.
>
> Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police***.

Example

Example 1:

```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

Idea

💡 Keep track of the max of all previous 0 to i-2 element, add to the current value to make it the largest. Keep track of a global max for output

Solution

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        rob1, rob2 = 0, 0  # Initialize two variables to represent the maximum amount robbed up to two houses ago and one house ago.

        # [rob1, rob2, n, n+1]
        for n in nums:
            temp = max(n + rob1, rob2)  # Calculate the maximum amount that can be robbed considering the current house.
            rob1 = rob2  # Update rob1 to represent the maximum amount robbed one house ago.
            rob2 = temp  # Update rob2 to represent the maximum amount robbed up to the current house.
        return rob2  # Return the maximum amount that can be robbed from all the houses.
```

> **Explanation:**
>
> 1. The `rob` method takes a list `nums` as input, where `nums` represents the amount of money that can be robbed from each house. The goal is to determine the maximum amount that can be robbed without robbing adjacent houses.
> 2. Two variables, `rob1` and `rob2`, are initialized to 0. These variables represent the maximum amount that can be robbed up to two houses ago and one house ago, respectively.
> 3. The code uses a `for` loop to iterate through the `nums` list, representing the amount of money in each house.
> 4. Inside the loop, the `temp` variable is calculated using the `max` function. It represents the maximum amount that can be robbed considering the current house. There are two options to consider:
>   - Robbing the current house (`n + rob1`): This is the maximum amount if the current house is robbed, and the maximum amount robbed up to two houses ago (`rob1`).
>   - Not robbing the current house (`rob2`): This is the maximum amount if the current house is not robbed, and the maximum amount robbed up to one house ago (`rob2`).
> 5. `rob1` is then updated to represent the maximum amount robbed one house ago, and `rob2` is updated to represent the maximum amount robbed up to the current house. These updates prepare for the next iteration.
> 6. The loop continues through all the houses in `nums`, and at the end, `rob2` contains the maximum amount that can be robbed from all the houses without robbing adjacent houses.
> 7. Finally, the method returns `rob2`, which is the answer to the problem, representing the maximum amount of money that can be robbed.

> Certainly, I'll walk you through the code with a step-by-step "dry run" (step-by-step execution) using a simple example:

> Let's say we have the following input `nums` list, which represents the amount of money in each house:
>
> ```
> nums = [2, 7, 9, 3, 1]
> ```
>
> We want to determine the maximum amount of money that can be robbed from these houses without robbing adjacent houses.
>
> 1. Initialize `rob1` and `rob2` to 0. These variables represent the maximum amount robbed up to two houses ago and one house ago, respectively.
>
>    ```
>    rob1 = 0
>    rob2 = 0
>    ```
>
> 2. Enter the `for` loop to iterate through the `nums` list.
>
>   - For the first house (2), calculate `temp`:
>
>      ```
>      temp = max(2 + 0, 0) = max(2, 0) = 2
>      ```
>
>   Update `rob1` and `rob2`:
>
>      ```
>      rob1 = 0
>      rob2 = 2
>      ```
>
>   - For the second house (7), calculate `temp`:
>
>      ```
>      temp = max(7 + 0, 2) = max(7, 2) = 7
>      ```
>
>   Update `rob1` and `rob2`:
>
>      ```
>      rob1 = 2
>      rob2 = 7
>      ```
>
>   - For the third house (9), calculate `temp`:
>
>      ```
>      temp = max(9 + 2, 7) = max(11, 7) = 11
>      ```
>
>   Update `rob1` and `rob2`:

```
    rob1 = 7
    rob2 = 11
    ```

  - For the fourth house (3), calculate `temp`:

    ```
    temp = max(3 + 7, 11) = max(10, 11) = 11
    ```

    Update `rob1` and `rob2`:

    ```
    rob1 = 11
    rob2 = 11
    ```

  - For the fifth house (1), calculate `temp`:

    ```
    temp = max(1 + 11, 11) = max(12, 11) = 12
    ```

    Update `rob1` and `rob2`:

    ```
    rob1 = 11
    rob2 = 12
    ```

3. The loop has completed, and `rob2` contains the maximum amount that can be robbed without robbing adjacent houses, which is 12.
4. Return `rob2` as the final result, which is the maximum amount that can be robbed:

   ```
   return 12
   ```

So, in this example, the maximum amount that can be robbed from the houses `[2, 7, 9, 3, 1]` without robbing adjacent houses is 12.