```swift
import Foundation
/// ** 剑指 Offer 60. n 个骰子的点数**
/// 把 n 个骰子扔在地上，所有骰子朝上一面的点数之和为 s 。
/// 输入 n，打印出 s 的所有可能的值出现的概率。
///
/// 你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能
/// 掷出的点数集合中第 i 小的那个的概率。
class Solution {
    /// 利用动态规划设已知 n-1 个骰子的概率求添加一个骰子 n 个骰子的概率分布
    /// 不论添加的一个骰子的点数 i 为多少，设点数之和为 x，只要前 n-1 个骰子
    /// 之和为 x-i 即可得到点数之和为 x 的概率
    /// **举例：**
    /// 记录一个骰子的结果为[0.16667, 0.16667, 0.16667, 0.16667, 0.16667, 0.16667]
    /// 则两个骰子的求和结果范围为 2～12
    /// 以总点数为2举例，当总点数和为 x = 2 时，
    ///     - 新加入的骰子点数为 1 此时概率为 P(前 n-1 个骰子之和为1) ✖ P(新加入骰子点数为1)
    ///     - 新加入的骰子点数为 2 此时概率为 P(前 n-1 个骰子之和为0) ✖ P(新加入骰子点数为2)
    ///     ...
    ///     - 新加入的骰子点数为 6 此时概率为 P(前 n-1 个骰子之和为-4) ✖ P(新加入骰子点数为6)
    /// 将上述所有概率相加即可得到 2 个骰子总点数和为 x = 2 时的概率
    func dicesProbability(_ n: Int) -> [Double] {
        /// 一个骰子的概率情况
        var dp: [Double] = Array(repeating: 1/6, count: 6)
        if n == 1 { return dp }
        /// 遍历多次不断迭代生成新的DP
        for index in 2...n {
            var (dpCount, nextDP): (Int, [Double]) = (dp.count - 1, [])
            /// 总骰子点数为 n～6n
            for total in index...6*index {
                /// 计算总点数为 total 时候的概率
                var probability: Double = 0.0
                /// 新加入的骰子的点数取值范围为1～6
                for i in 1...6 {
                    /// 每次迭代之后 dp 内最小点数会发生变化因此索引的位置不是
                    /// 单纯的 total - i - 1
                    if ((total - i - (index-1) >= 0) &&
                        (total - i - (index-1) <= dpCount)) {
                        probability += dp[total-i-(index-1)]*(1/6)
                    }
                }
                nextDP.append(probability)
            }
            dp = nextDP
        }
        return dp
    }
}
```

```
let solution = Solution()
print(solution.dicesProbability(1))
print(solution.dicesProbability(2))
print(solution.dicesProbability(3))
```