Problem 1716: Calculate Money in Leetcode Bank

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Hercy wants to save money for his first car. He puts money in the Leetcode bank

every day

.

He starts by putting in

$1

on Monday, the first day. Every day from Tuesday to Sunday, he will put in

$1

more than the day before. On every subsequent Monday, he will put in

$1

more than the

previous Monday

.

Given

n

, return

the total amount of money he will have in the Leetcode bank at the end of the

n

th

day.

Example 1:

Input:

n = 4

Output:

10

Explanation:

After the 4

th

day, the total is 1 + 2 + 3 + 4 = 10.

Example 2:

Input:

n = 10

Output:

37

Explanation:

After the 10

th

day, the total is (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4) = 37. Notice that on the 2

nd

Monday, Hercy only puts in $2.

Example 3:

Input:

n = 20

Output:

96

Explanation:

After the 20

th

day, the total is (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4 + 5 + 6 + 7 + 8) + (3 + 4 + 5 + 6 + 7 + 8) = 96.

Constraints:

1 <= n <= 1000

Code Snippets

C++:

```
class Solution {
public:
int totalMoney(int n) {

}
};
```

Java:

```
class Solution {
public int totalMoney(int n) {

}
}
```

Python3:

```
class Solution:
def totalMoney(self, n: int) -> int:
```

Python:

```
class Solution(object):
def totalMoney(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var totalMoney = function(n) {

};
```

TypeScript:

```
function totalMoney(n: number): number {

};
```

C#:

```
public class Solution {
public int TotalMoney(int n) {

}
}
```

C:

```
int totalMoney(int n) {

}
```

Go:

```
func totalMoney(n int) int {

}
```

Kotlin:

```
class Solution {
fun totalMoney(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func totalMoney(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn total_money(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def total_money(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function totalMoney($n) {

}
}
```

Dart:

```
class Solution {
int totalMoney(int n) {

}
}
```

Scala:

```
object Solution {
def totalMoney(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec total_money(n :: integer) :: integer
def total_money(n) do

end
end
```

Erlang:

```
-spec total_money(N :: integer()) -> integer().
total_money(N) ->
.
```

Racket:

```
(define/contract (total-money n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Calculate Money in Leetcode Bank
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int totalMoney(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Calculate Money in Leetcode Bank
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int totalMoney(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Calculate Money in Leetcode Bank
Difficulty: Easy
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def totalMoney(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def totalMoney(self, n):
"""
:type n: int
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Calculate Money in Leetcode Bank
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {number}
*/
var totalMoney = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Calculate Money in Leetcode Bank
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function totalMoney(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Calculate Money in Leetcode Bank
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int TotalMoney(int n) {

}
}
```

C Solution:

```
/*
* Problem: Calculate Money in Leetcode Bank
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int totalMoney(int n) {

}
```

Go Solution:

```
// Problem: Calculate Money in Leetcode Bank
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach
```

```
func totalMoney(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun totalMoney(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func totalMoney(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Calculate Money in Leetcode Bank
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn total_money(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def total_money(n)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function totalMoney($n) {

}
}
```

Dart Solution:

```
class Solution {
int totalMoney(int n) {

}
}
```

Scala Solution:

```
object Solution {
def totalMoney(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec total_money(n :: integer) :: integer
def total_money(n) do

end
end
```

Erlang Solution:

```
-spec total_money(N :: integer()) -> integer().
total_money(N) ->
.
```

Racket Solution:

```
(define/contract (total-money n)
(-> exact-integer? exact-integer?)
)
```