Problem 2652: Sum Multiples

Problem Information

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

Problem Description

Given a positive integer

n

, find the sum of all integers in the range

[1, n]

inclusive

that are divisible by

3

,

5

, or

7

.

Return

an integer denoting the sum of all numbers in the given range satisfying the constraint.

Example 1:

Input:

n = 7

Output:

21

Explanation:

Numbers in the range

[1, 7]

that are divisible by

3

,

5,

or

7

are

3, 5, 6, 7

. The sum of these numbers is

21

.

Example 2:

Input:

n = 10

Output:

40

Explanation:

Numbers in the range

[1, 10] that are

divisible by

3

,

5,

or

7

are

3, 5, 6, 7, 9, 10

. The sum of these numbers is 40.

Example 3:

Input:

n = 9

Output:

30

Explanation:

Numbers in the range

[1, 9]

that are divisible by

3

,

5

, or

7

are

3, 5, 6, 7, 9

. The sum of these numbers is

30

.

Constraints:

1 <= n <= 10

3

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int sumOfMultiples(int n) {

}
```

Go:

```
func sumOfMultiples(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}

}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-multiples n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum Multiples
* 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 sumOfMultiples(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Sum Multiples
* 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 sumOfMultiples(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum Multiples
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 sumOfMultiples(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum Multiples
* 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 sumOfMultiples = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum Multiples
* 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 sumOfMultiples(n: number): number {
};
```

C# Solution:

```
/*
* Problem: Sum Multiples
* 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 SumOfMultiples(int n) {

}
}
```

C Solution:

```
/*
* Problem: Sum Multiples
* 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 sumOfMultiples(int n) {

}
```

Go Solution:

```
// Problem: Sum Multiples
// 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 sumOfMultiples(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum Multiples
// 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 sum_of_multiples(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-multiples n)
(-> exact-integer? exact-integer?)
)
```