Problem 1281: Subtract the Product and Sum of Digits of an Integer

Problem Information

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

Problem Description

Given an integer number

n

, return the difference between the product of its digits and the sum of its digits.

Example 1:

Input:

n = 234

Output:

15

Explanation:

Product of digits = 2 * 3 * 4 = 24 Sum of digits = 2 + 3 + 4 = 9 Result = 24 - 9 = 15

Example 2:

Input:

n = 4421

Output:

21

Explanation:

Product of digits = 4 * 4 * 2 * 1 = 32 Sum of digits = 4 + 4 + 2 + 1 = 11 Result = 32 - 11 = 21

Constraints:

1 <= n <= 10^5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def subtractProductAndSum(self, n):
```

```
"""
:type n: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int subtractProductAndSum(int n) {

}
```

Go:

```
func subtractProductAndSum(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (subtract-product-and-sum n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subtract the Product and Sum of Digits of an Integer
* 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 subtractProductAndSum(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Subtract the Product and Sum of Digits of an Integer
* 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 subtractProductAndSum(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Subtract the Product and Sum of Digits of an Integer
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 subtractProductAndSum(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Subtract the Product and Sum of Digits of an Integer
* 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 subtractProductAndSum = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Subtract the Product and Sum of Digits of an Integer
* 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 subtractProductAndSum(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Subtract the Product and Sum of Digits of an Integer
* 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 SubtractProductAndSum(int n) {

}
}
```

C Solution:

```
/*
* Problem: Subtract the Product and Sum of Digits of an Integer
* 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 subtractProductAndSum(int n) {

}
```

Go Solution:

```
// Problem: Subtract the Product and Sum of Digits of an Integer
// 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 subtractProductAndSum(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Subtract the Product and Sum of Digits of an Integer
// 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 subtract_product_and_sum(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (subtract-product-and-sum n)
(-> exact-integer? exact-integer?)
)
```