Problem 2544: Alternating Digit Sum

Problem Information

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

Problem Description

You are given a positive integer

n

. Each digit of

n

has a sign according to the following rules:

The

most significant digit

is assigned a

positive

sign.

Each other digit has an opposite sign to its adjacent digits.

Return

the sum of all digits with their corresponding sign

.

Example 1:

Input:

n = 521

Output:

4

Explanation:

(+5) + (-2) + (+1) = 4.

Example 2:

Input:

n = 111

Output:

1

Explanation:

(+1) + (-1) + (+1) = 1.

Example 3:

Input:

n = 886996

Output:

0

Explanation:

(+8) + (-8) + (+6) + (-9) + (+9) + (-6) = 0.

Constraints:

1 <= n <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int alternateDigitSum(int n) {

}
```

Go:

```
func alternateDigitSum(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (alternate-digit-sum n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Alternating Digit Sum
* 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 alternateDigitSum(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Alternating Digit Sum
* 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 alternateDigitSum(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Alternating Digit Sum
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 alternateDigitSum(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Alternating Digit Sum
* 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 alternateDigitSum = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Alternating Digit Sum
* 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 alternateDigitSum(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Alternating Digit Sum
* 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 AlternateDigitSum(int n) {

}
}
```

C Solution:

```
/*
* Problem: Alternating Digit Sum
* 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 alternateDigitSum(int n) {

}
```

Go Solution:

```
// Problem: Alternating Digit Sum
// 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 alternateDigitSum(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Alternating Digit Sum
// 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 alternate_digit_sum(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (alternate-digit-sum n)
(-> exact-integer? exact-integer?)
)
```