Problem 2520: Count the Digits That Divide a Number

Problem Information

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

Problem Description

Given an integer

num

, return

the number of digits in

num

that divide

num

.

An integer

val

divides

nums

if

nums % val == 0

.

Example 1:

Input:

num = 7

Output:

1

Explanation:

7 divides itself, hence the answer is 1.

Example 2:

Input:

num = 121

Output:

2

Explanation:

121 is divisible by 1, but not 2. Since 1 occurs twice as a digit, we return 2.

Example 3:

Input:

num = 1248

Output:

4

Explanation:

1248 is divisible by all of its digits, hence the answer is 4.

Constraints:

1 <= num <= 10

9

num

does not contain

0

as one of its digits.

Code Snippets

C++:

```
class Solution {
public:
int countDigits(int num) {

}
};
```

Java:

```
class Solution {
public int countDigits(int num) {

}
```

```
}
```

Python3:

```
class Solution:
def countDigits(self, num: int) -> int:
```

Python:

```
class Solution(object):
def countDigits(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} num
* @return {number}
*/
var countDigits = function(num) {

};
```

TypeScript:

```
function countDigits(num: number): number {

};
```

C#:

```
public class Solution {
public int CountDigits(int num) {

}
}
```

C:

```
int countDigits(int num) {

}
```

Go:

```
func countDigits(num int) int {

}
```

Kotlin:

```
class Solution {
fun countDigits(num: Int): Int {

}
}
```

Swift:

```
class Solution {
func countDigits(_ num: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_digits(num: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num
# @return {Integer}
def count_digits(num)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function countDigits($num) {

}
}
```

Dart:

```
class Solution {
int countDigits(int num) {

}
}
```

Scala:

```
object Solution {
def countDigits(num: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_digits(num :: integer) :: integer
def count_digits(num) do

end
end
```

Erlang:

```
-spec count_digits(Num :: integer()) -> integer().
count_digits(Num) ->
.
```

Racket:

```
(define/contract (count-digits num)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Digits That Divide a Number
* 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 countDigits(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Digits That Divide a Number
* 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 countDigits(int num) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count the Digits That Divide a Number
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 countDigits(self, num: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countDigits(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count the Digits That Divide a Number
* 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} num
* @return {number}
*/
var countDigits = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Digits That Divide a Number
* 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 countDigits(num: number): number {

};
```

C# Solution:

```
/*
* Problem: Count the Digits That Divide a Number
* 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 CountDigits(int num) {

}
}
```

C Solution:

```
/*
* Problem: Count the Digits That Divide a Number
* 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 countDigits(int num) {

}
```

Go Solution:

```
// Problem: Count the Digits That Divide a Number
// 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 countDigits(num int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countDigits(num: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countDigits(_ num: Int) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Count the Digits That Divide a Number
// 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 count_digits(num: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer}
def count_digits(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function countDigits($num) {

}
}
```

Dart Solution:

```
class Solution {
int countDigits(int num) {

}
}
```

Scala Solution:

```
object Solution {
def countDigits(num: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_digits(num :: integer) :: integer
def count_digits(num) do

end
end
```

Erlang Solution:

```
-spec count_digits(Num :: integer()) -> integer().
count_digits(Num) ->
.
```

Racket Solution:

```
(define/contract (count-digits num)
(-> exact-integer? exact-integer?)
)
```