Problem 1837: Sum of Digits in Base K

Problem Information

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

Problem Description

Given an integer

n

(in base

10

) and a base

k

, return

the

sum

of the digits of

n

after

converting

n

from base

10

to base

k

.

After converting, each digit should be interpreted as a base

10

number, and the sum should be returned in base

10

.

Example 1:

Input:

n = 34, k = 6

Output:

9

Explanation:

34 (base 10) expressed in base 6 is 54. 5 + 4 = 9.

Example 2:

Input:

n = 10, k = 10

Output:

1

Explanation:

n is already in base 10. 1 + 0 = 1.

Constraints:

1 <= n <= 100

2 <= k <= 10

Code Snippets

C++:

```
class Solution {
public:
int sumBase(int n, int k) {

}
};
```

Java:

```
class Solution {
public int sumBase(int n, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var sumBase = function(n, k) {

};
```

TypeScript:

```
function sumBase(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int SumBase(int n, int k) {

}
}
```

C:

```
int sumBase(int n, int k) {

}
```

Go:

```
func sumBase(n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun sumBase(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func sumBase(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn sum_base(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def sum_base(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
```

```
* @param Integer $k
* @return Integer
*/
function sumBase($n, $k) {

}
}
```

Dart:

```
class Solution {
int sumBase(int n, int k) {

}
}
```

Scala:

```
object Solution {
def sumBase(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_base(n :: integer, k :: integer) :: integer
def sum_base(n, k) do

end
end
```

Erlang:

```
-spec sum_base(N :: integer(), K :: integer()) -> integer().
sum_base(N, K) ->
.
```

Racket:

```
(define/contract (sum-base n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Digits in Base K
* 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 sumBase(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Digits in Base K
* 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 sumBase(int n, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Sum of Digits in Base K
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 sumBase(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Digits in Base K
* 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
* @param {number} k
* @return {number}
*/
var sumBase = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Digits in Base K
* 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 sumBase(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Digits in Base K
* 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 SumBase(int n, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Sum of Digits in Base K
* 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 sumBase(int n, int k) {

}
```

Go Solution:

```
// Problem: Sum of Digits in Base K
// 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 sumBase(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sumBase(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func sumBase(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Digits in Base K
// 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_base(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def sum_base(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function sumBase($n, $k) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int sumBase(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def sumBase(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_base(n :: integer, k :: integer) :: integer
def sum_base(n, k) do

end
end
```

Erlang Solution:

```
-spec sum_base(N :: integer(), K :: integer()) -> integer().
sum_base(N, K) ->
.
```

Racket Solution:

```
(define/contract (sum-base n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```