# Problem 3483: Unique 3-Digit Even Numbers

## Problem Information

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

## Problem Description

You are given an array of digits called

digits

. Your task is to determine the number of

distinct

three-digit even numbers that can be formed using these digits.

Note

: Each

copy

of a digit can only be used

once per number

, and there may

not

be leading zeros.

Example 1:

Input:

digits = [1,2,3,4]

Output:

12

Explanation:

The 12 distinct 3-digit even numbers that can be formed are 124, 132, 134, 142, 214, 234, 312, 314, 324, 342, 412, and 432. Note that 222 cannot be formed because there is only 1 copy of the digit 2.

Example 2:

Input:

digits = [0,2,2]

Output:

2

Explanation:

The only 3-digit even numbers that can be formed are 202 and 220. Note that the digit 2 can be used twice because it appears twice in the array.

Example 3:

Input:

digits = [6,6,6]

Output:

1

Explanation:

Only 666 can be formed.

Example 4:

Input:

digits = [1,3,5]

Output:

0

Explanation:

No even 3-digit numbers can be formed.

Constraints:

3 <= digits.length <= 10

0 <= digits[i] <= 9

## Code Snippets

**C++:**

```
class Solution {
public:
int totalNumbers(vector<int>& digits) {

}
};
```

**Java:**

```
class Solution {
public int totalNumbers(int[] digits) {

}
}
```

**Python3:**

```
class Solution:
def totalNumbers(self, digits: List[int]) -> int:
```

**Python:**

```
class Solution(object):
def totalNumbers(self, digits):
"""
:type digits: List[int]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} digits
* @return {number}
*/
var totalNumbers = function(digits) {

};
```

**TypeScript:**

```
function totalNumbers(digits: number[]): number {

};
```

**C#:**

```
public class Solution {
public int TotalNumbers(int[] digits) {

}
}
```

**C:**

```
int totalNumbers(int* digits, int digitsSize) {

}
```

**Go:**

```
func totalNumbers(digits []int) int {

}
```

**Kotlin:**

```
class Solution {
fun totalNumbers(digits: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func totalNumbers(_ digits: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn total_numbers(digits: Vec<i32>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} digits
# @return {Integer}
def total_numbers(digits)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $digits
* @return Integer
*/
function totalNumbers($digits) {

}
}
```

**Dart:**

```
class Solution {
int totalNumbers(List<int> digits) {

}
}
```

**Scala:**

```
object Solution {
def totalNumbers(digits: Array[Int]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec total_numbers(digits :: [integer]) :: integer
def total_numbers(digits) do

end
end
```

**Erlang:**

```
-spec total_numbers(Digits :: [integer()]) -> integer().
total_numbers(Digits) ->
.
```

### Racket:

```
(define/contract (total-numbers digits)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Unique 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int totalNumbers(vector<int>& digits) {

}
};
```

### Java Solution:

```
/**
* Problem: Unique 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int totalNumbers(int[] digits) {
```

```
}
}
```

### Python3 Solution:

```
"""
Problem: Unique 3-Digit Even Numbers
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def totalNumbers(self, digits: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def totalNumbers(self, digits):
"""
:type digits: List[int]
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Unique 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* @param {number[]} digits
* @return {number}
*/
var totalNumbers = function(digits) {

};
```

### TypeScript Solution:

```
/**
* Problem: Unique 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function totalNumbers(digits: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Unique 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int TotalNumbers(int[] digits) {

}
```

```
}
```

### C Solution:

```
/*
* Problem: Unique 3-Digit Even Numbers
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int totalNumbers(int* digits, int digitsSize) {

}
```

### Go Solution:

```
// Problem: Unique 3-Digit Even Numbers
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func totalNumbers(digits []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun totalNumbers(digits: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func totalNumbers(_ digits: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Unique 3-Digit Even Numbers
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn total_numbers(digits: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} digits
# @return {Integer}
def total_numbers(digits)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $digits
* @return Integer
*/
function totalNumbers($digits) {

}
}
```

### Dart Solution:

```
class Solution {
int totalNumbers(List<int> digits) {

}
}
```

### Scala Solution:

```
object Solution {
def totalNumbers(digits: Array[Int]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec total_numbers(digits :: [integer]) :: integer
def total_numbers(digits) do

end
end
```

### Erlang Solution:

```
-spec total_numbers(Digits :: [integer()]) -> integer().
total_numbers(Digits) ->
.
```

### Racket Solution:

```
(define/contract (total-numbers digits)
(-> (listof exact-integer?) exact-integer?)
)
```