

Problem 2235: Add Two Integers

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given two integers

num1

and

num2

, return

the

sum

of the two integers

.

Example 1:

Input:

num1 = 12, num2 = 5

Output:

17

Explanation:

num1 is 12, num2 is 5, and their sum is $12 + 5 = 17$, so 17 is returned.

Example 2:

Input:

num1 = -10, num2 = 4

Output:

-6

Explanation:

$\text{num1} + \text{num2} = -6$, so -6 is returned.

Constraints:

$-100 \leq \text{num1}, \text{num2} \leq 100$

Code Snippets

C++:

```
class Solution {  
public:  
    int sum(int num1, int num2) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int sum(int num1, int num2) {
```

```
}  
}
```

Python3:

```
class Solution:  
    def sum(self, num1: int, num2: int) -> int:
```

Python:

```
class Solution(object):  
    def sum(self, num1, num2):  
        """  
        :type num1: int  
        :type num2: int  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number} num1  
 * @param {number} num2  
 * @return {number}  
 */  
var sum = function(num1, num2) {  
  
};
```

TypeScript:

```
function sum(num1: number, num2: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int Sum(int num1, int num2) {  
  
    }  
}
```

```
}
```

C:

```
int sum(int num1, int num2) {  
  
}
```

Go:

```
func sum(num1 int, num2 int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun sum(num1: Int, num2: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func sum(_ num1: Int, _ num2: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn sum(num1: i32, num2: i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer} num1  
# @param {Integer} num2
```

```
# @return {Integer}
def sum(num1, num2)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer $num1
     * @param Integer $num2
     * @return Integer
     */
    function sum($num1, $num2) {

    }

}
```

Dart:

```
class Solution {
  int sum(int num1, int num2) {

  }

}
```

Scala:

```
object Solution {
  def sum(num1: Int, num2: Int): Int = {

  }

}
```

Elixir:

```
defmodule Solution do
  @spec sum(num1 :: integer, num2 :: integer) :: integer
  def sum(num1, num2) do

  end
end
```

```
end
```

Erlang:

```
-spec sum(Num1 :: integer(), Num2 :: integer()) -> integer().  
sum(Num1, Num2) ->  
.
```

Racket:

```
(define/contract (sum num1 num2)  
  (-> exact-integer? exact-integer? exact-integer?)  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Add Two Integers  
 * 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 sum(int num1, int num2) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Add Two Integers  
 * 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 sum(int num1, int num2) {

}
}

```

Python3 Solution:

```

"""
Problem: Add Two Integers
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 sum(self, num1: int, num2: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def sum(self, num1, num2):
        """
        :type num1: int
        :type num2: int
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Add Two Integers
 * 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} num1
 * @param {number} num2
 * @return {number}
 */
var sum = function(num1, num2) {

};

```

TypeScript Solution:

```

/**
 * Problem: Add Two Integers
 * 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 sum(num1: number, num2: number): number {

};

```

C# Solution:

```

/*
 * Problem: Add Two Integers
 * 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 Sum(int num1, int num2) {

}

}

```

C Solution:

```

/*
* Problem: Add Two Integers
* 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 sum(int num1, int num2) {

}

```

Go Solution:

```

// Problem: Add Two Integers
// 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 sum(num1 int, num2 int) int {

}

```

Kotlin Solution:

```
class Solution {  
    fun sum(num1: Int, num2: Int): Int {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func sum(_ num1: Int, _ num2: Int) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Add Two Integers  
// 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(num1: i32, num2: i32) -> i32 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer} num1  
# @param {Integer} num2  
# @return {Integer}  
def sum(num1, num2)  
  
end
```

PHP Solution:

```

class Solution {

  /**
   * @param Integer $num1
   * @param Integer $num2
   * @return Integer
   */
  function sum($num1, $num2) {

  }

}

```

Dart Solution:

```

class Solution {
  int sum(int num1, int num2) {

  }

}

```

Scala Solution:

```

object Solution {
  def sum(num1: Int, num2: Int): Int = {

  }

}

```

Elixir Solution:

```

defmodule Solution do
  @spec sum(num1 :: integer, num2 :: integer) :: integer
  def sum(num1, num2) do

  end

end

```

Erlang Solution:

```

-spec sum(Num1 :: integer(), Num2 :: integer()) -> integer().
sum(Num1, Num2) ->

.

```

Racket Solution:

```
(define/contract (sum num1 num2)
  (-> exact-integer? exact-integer? exact-integer?)
  )
```