Problem 3222: Find the Winning Player in Coin Game

Problem Information

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

Problem Description

You are given two

positive

integers

x

and

y

, denoting the number of coins with values 75 and 10

respectively

.

Alice and Bob are playing a game. Each turn, starting with

Alice

, the player must pick up coins with a

total

value 115. If the player is unable to do so, they

lose

the game.

Return the

name

of the player who wins the game if both players play

optimally

.

Example 1:

Input:

x = 2, y = 7

Output:

"Alice"

Explanation:

The game ends in a single turn:

Alice picks 1 coin with a value of 75 and 4 coins with a value of 10.

Example 2:

Input:

x = 4, y = 11

Output:

"Bob"

Explanation:

The game ends in 2 turns:

Alice picks 1 coin with a value of 75 and 4 coins with a value of 10.

Bob picks 1 coin with a value of 75 and 4 coins with a value of 10.

Constraints:

1 <= x, y <= 100

Code Snippets

C++:

```
class Solution {
public:
string winningPlayer(int x, int y) {

}
};
```

Java:

```
class Solution {
public String winningPlayer(int x, int y) {

}
}
```

Python3:

```
class Solution:
def winningPlayer(self, x: int, y: int) -> str:
```

Python:

```
class Solution(object):
def winningPlayer(self, x, y):
"""
:type x: int
:type y: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @return {string}
*/
var winningPlayer = function(x, y) {

};
```

TypeScript:

```
function winningPlayer(x: number, y: number): string {

};
```

C#:

```
public class Solution {
public string WinningPlayer(int x, int y) {

}
}
```

C:

```
char* winningPlayer(int x, int y) {

}
```

Go:

```
func winningPlayer(x int, y int) string {

}
```

Kotlin:

```
class Solution {
fun winningPlayer(x: Int, y: Int): String {

}
}
```

Swift:

```
class Solution {
func winningPlayer(_ x: Int, _ y: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn winning_player(x: i32, y: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @return {String}
def winning_player(x, y)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
```

```
* @param Integer $y
* @return String
*/
function winningPlayer($x, $y) {

}
}
```

Dart:

```
class Solution {
String winningPlayer(int x, int y) {

}
}
```

Scala:

```
object Solution {
def winningPlayer(x: Int, y: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec winning_player(x :: integer, y :: integer) :: String.t
def winning_player(x, y) do

end
end
```

Erlang:

```
-spec winning_player(X :: integer(), Y :: integer()) ->
unicode:unicode_binary().
winning_player(X, Y) ->
.
```

Racket:

```
(define/contract (winning-player x y)
(-> exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Winning Player in Coin Game
* 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:
string winningPlayer(int x, int y) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Winning Player in Coin Game
* 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 String winningPlayer(int x, int y) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Find the Winning Player in Coin Game
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 winningPlayer(self, x: int, y: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def winningPlayer(self, x, y):
"""
:type x: int
:type y: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Winning Player in Coin Game
* 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} x
* @param {number} y
* @return {string}
*/
var winningPlayer = function(x, y) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Winning Player in Coin Game
* 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 winningPlayer(x: number, y: number): string {

};
```

C# Solution:

```
/*
* Problem: Find the Winning Player in Coin Game
* 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 string WinningPlayer(int x, int y) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Find the Winning Player in Coin Game
* 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
*/

char* winningPlayer(int x, int y) {

}
```

Go Solution:

```
// Problem: Find the Winning Player in Coin Game
// 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 winningPlayer(x int, y int) string {

}
```

Kotlin Solution:

```
class Solution {
fun winningPlayer(x: Int, y: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func winningPlayer(_ x: Int, _ y: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Find the Winning Player in Coin Game
// 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 winning_player(x: i32, y: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @return {String}
def winning_player(x, y)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @return String
*/
function winningPlayer($x, $y) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String winningPlayer(int x, int y) {

}
}
```

Scala Solution:

```
object Solution {
def winningPlayer(x: Int, y: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec winning_player(x :: integer, y :: integer) :: String.t
def winning_player(x, y) do

end
end
```

Erlang Solution:

```
-spec winning_player(X :: integer(), Y :: integer()) ->
unicode:unicode_binary().
winning_player(X, Y) ->
.
```

Racket Solution:

```
(define/contract (winning-player x y)
(-> exact-integer? exact-integer? string?)
)
```