Problem 2806: Account Balance After Rounded Purchase

Problem Information

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

Problem Description

Initially, you have a bank account balance of

100

dollars.

You are given an integer

purchaseAmount

representing the amount you will spend on a purchase in dollars, in other words, its price.

When making the purchase, first the

purchaseAmount

is rounded to the nearest multiple of 10

. Let us call this value

roundedAmount

. Then,

roundedAmount

dollars are removed from your bank account.

Return an integer denoting your final bank account balance after this purchase.

Notes:

0 is considered to be a multiple of 10 in this problem.

When rounding, 5 is rounded upward (5 is rounded to 10, 15 is rounded to 20, 25 to 30, and so on).

Example 1:

Input:

purchaseAmount = 9

Output:

90

Explanation:

The nearest multiple of 10 to 9 is 10. So your account balance becomes 100 - 10 = 90.

Example 2:

Input:

purchaseAmount = 15

Output:

80

Explanation:

The nearest multiple of 10 to 15 is 20. So your account balance becomes 100 - 20 = 80.

Example 3:

Input:

purchaseAmount = 10

Output:

90

Explanation:

10 is a multiple of 10 itself. So your account balance becomes 100 - 10 = 90.

Constraints:

0 <= purchaseAmount <= 100

Code Snippets

C++:

```
class Solution {
public:
int accountBalanceAfterPurchase(int purchaseAmount) {

}
};
```

Java:

```
class Solution {
public int accountBalanceAfterPurchase(int purchaseAmount) {

}
}
```

Python3:

```
class Solution:
def accountBalanceAfterPurchase(self, purchaseAmount: int) -> int:
```

Python:

```
class Solution(object):
def accountBalanceAfterPurchase(self, purchaseAmount):
"""
:type purchaseAmount: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} purchaseAmount
* @return {number}
*/
var accountBalanceAfterPurchase = function(purchaseAmount) {

};
```

TypeScript:

```
function accountBalanceAfterPurchase(purchaseAmount: number): number {

};
```

C#:

```
public class Solution {
public int AccountBalanceAfterPurchase(int purchaseAmount) {

}
}
```

C:

```
int accountBalanceAfterPurchase(int purchaseAmount) {

}
```

Go:

```
func accountBalanceAfterPurchase(purchaseAmount int) int {

}
```

Kotlin:

```
class Solution {
fun accountBalanceAfterPurchase(purchaseAmount: Int): Int {

}
}
```

Swift:

```
class Solution {
func accountBalanceAfterPurchase(_ purchaseAmount: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn account_balance_after_purchase(purchase_amount: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} purchase_amount
# @return {Integer}
def account_balance_after_purchase(purchase_amount)

end
```

PHP:

```
class Solution {

/**
* @param Integer $purchaseAmount
* @return Integer
*/
function accountBalanceAfterPurchase($purchaseAmount) {

}
}
```

Dart:

```
class Solution {
int accountBalanceAfterPurchase(int purchaseAmount) {

}
}
```

Scala:

```
object Solution {
def accountBalanceAfterPurchase(purchaseAmount: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec account_balance_after_purchase(purchase_amount :: integer) :: integer
def account_balance_after_purchase(purchase_amount) do

end
end
```

Erlang:

```
-spec account_balance_after_purchase(PurchaseAmount :: integer()) ->
integer().
account_balance_after_purchase(PurchaseAmount) ->
.
```

Racket:

```
(define/contract (account-balance-after-purchase purchaseAmount)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Account Balance After Rounded Purchase
* 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 accountBalanceAfterPurchase(int purchaseAmount) {

}
};
```

Java Solution:

```
/**
* Problem: Account Balance After Rounded Purchase
* 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 accountBalanceAfterPurchase(int purchaseAmount) {

}
}
```

Python3 Solution:

```
"""
Problem: Account Balance After Rounded Purchase
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 accountBalanceAfterPurchase(self, purchaseAmount: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def accountBalanceAfterPurchase(self, purchaseAmount):
"""
:type purchaseAmount: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Account Balance After Rounded Purchase
* 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} purchaseAmount
* @return {number}
*/
```

```
var accountBalanceAfterPurchase = function(purchaseAmount) {

};
```

TypeScript Solution:

```
/**
* Problem: Account Balance After Rounded Purchase
* 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 accountBalanceAfterPurchase(purchaseAmount: number): number {

};
```

C# Solution:

```
/*
* Problem: Account Balance After Rounded Purchase
* 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 AccountBalanceAfterPurchase(int purchaseAmount) {

}
}
```

C Solution:

```
/*
* Problem: Account Balance After Rounded Purchase
* 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 accountBalanceAfterPurchase(int purchaseAmount) {

}
```

Go Solution:

```
// Problem: Account Balance After Rounded Purchase
// 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 accountBalanceAfterPurchase(purchaseAmount int) int {

}
```

Kotlin Solution:

```
class Solution {
fun accountBalanceAfterPurchase(purchaseAmount: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func accountBalanceAfterPurchase(_ purchaseAmount: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Account Balance After Rounded Purchase
// 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 account_balance_after_purchase(purchase_amount: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} purchase_amount
# @return {Integer}
def account_balance_after_purchase(purchase_amount)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $purchaseAmount
* @return Integer
*/
function accountBalanceAfterPurchase($purchaseAmount) {

}
}
```

Dart Solution:

```
class Solution {
int accountBalanceAfterPurchase(int purchaseAmount) {

}
}
```

Scala Solution:

```
object Solution {
def accountBalanceAfterPurchase(purchaseAmount: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec account_balance_after_purchase(purchase_amount :: integer) :: integer
def account_balance_after_purchase(purchase_amount) do

end
end
```

Erlang Solution:

```
-spec account_balance_after_purchase(PurchaseAmount :: integer()) ->
integer().
account_balance_after_purchase(PurchaseAmount) ->
.
```

Racket Solution:

```
(define/contract (account-balance-after-purchase purchaseAmount)
(-> exact-integer? exact-integer?)
)
```