Problem 1244: Design A Leaderboard

Problem Information

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

Problem Description

Design a Leaderboard class, which has 3 functions:

addScore(playerId, score)

: Update the leaderboard by adding

score

to the given player's score. If there is no player with such id in the leaderboard, add him to the leaderboard with the given

score

.

top(K)

: Return the score sum of the top

K

players.

reset(playerId)

: Reset the score of the player with the given id to 0 (in other words erase it from the leaderboard). It is guaranteed that the player was added to the leaderboard before calling this function.

Initially, the leaderboard is empty.

Example 1:

Input:

["Leaderboard","addScore","addScore","addScore","addScore","addScore","top","reset","reset","addScore","top"] [[],[1,73],[2,56],[3,39],[4,51],[5,4],[1],[1],[2],[2,51],[3]]

Output:

[null,null,null,null,null,null,73,null,null,null,141]

Explanation:

Leaderboard leaderboard = new Leaderboard (); leaderboard.addScore(1,73); // leaderboard = [[1,73]]; leaderboard.addScore(2,56); // leaderboard = [[1,73],[2,56]]; leaderboard.addScore(3,39); // leaderboard = [[1,73],[2,56],[3,39]]; leaderboard.addScore(4,51); // leaderboard = [[1,73],[2,56],[3,39],[4,51]]; leaderboard.addScore(5,4); // leaderboard = [[1,73],[2,56],[3,39],[4,51],[5,4]]; leaderboard.top(1); // returns 73; leaderboard.reset(1); // leaderboard = [[2,56],[3,39],[4,51],[5,4]]; leaderboard.reset(2); // leaderboard = [[3,39],[4,51],[5,4]]; leaderboard.addScore(2,51); // leaderboard = [[2,51],[3,39],[4,51],[5,4]]; leaderboard.top(3); // returns 141 = 51 + 51 + 39;

Constraints:

1 <= playerId, K <= 10000

It's guaranteed that

K

is less than or equal to the current number of players.

1 <= score <= 100

There will be at most

1000

function calls.

Code Snippets

C++:

```
class Leaderboard {
public:
Leaderboard() {

}

void addScore(int playerId, int score) {

}

int top(int K) {

}

void reset(int playerId) {

}
};

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard* obj = new Leaderboard();
* obj->addScore(playerId,score);
* int param_2 = obj->top(K);
* obj->reset(playerId);
*/
```

Java:

```
class Leaderboard {

public Leaderboard() {

}

public void addScore(int playerId, int score) {

}

public int top(int K) {

}

public void reset(int playerId) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard obj = new Leaderboard();
* obj.addScore(playerId,score);
* int param_2 = obj.top(K);
* obj.reset(playerId);
*/
```

Python3:

```
class Leaderboard:

def __init__(self):


def addScore(self, playerId: int, score: int) -> None:


def top(self, K: int) -> int:


def reset(self, playerId: int) -> None:
```

```
# Your Leaderboard object will be instantiated and called as such:
# obj = Leaderboard()
# obj.addScore(playerId,score)
# param_2 = obj.top(K)
# obj.reset(playerId)
```

Python:

```
class Leaderboard(object):

def __init__(self):


def addScore(self, playerId, score):
"""
:type playerId: int
:type score: int
:rtype: None
"""


def top(self, K):
"""
:type K: int
:rtype: int
"""


def reset(self, playerId):
"""
:type playerId: int
:rtype: None
"""



# Your Leaderboard object will be instantiated and called as such:
# obj = Leaderboard()
# obj.addScore(playerId,score)
# param_2 = obj.top(K)
# obj.reset(playerId)
```

JavaScript:

```
var Leaderboard = function() {

};

/**
* @param {number} playerId
* @param {number} score
* @return {void}
*/
Leaderboard.prototype.addScore = function(playerId, score) {

};

/**
* @param {number} K
* @return {number}
*/
Leaderboard.prototype.top = function(K) {

};

/**
* @param {number} playerId
* @return {void}
*/
Leaderboard.prototype.reset = function(playerId) {

};

/**
* Your Leaderboard object will be instantiated and called as such:
* var obj = new Leaderboard()
* obj.addScore(playerId,score)
* var param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

TypeScript:

```
class Leaderboard {
constructor() {

}

addScore(playerId: number, score: number): void {

}

top(K: number): number {

}

reset(playerId: number): void {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* var obj = new Leaderboard()
* obj.addScore(playerId,score)
* var param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

C#:

```
public class Leaderboard {

public Leaderboard() {

}

public void AddScore(int playerId, int score) {

}

public int Top(int K) {

}

public void Reset(int playerId) {
```

```
}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard obj = new Leaderboard();
* obj.AddScore(playerId,score);
* int param_2 = obj.Top(K);
* obj.Reset(playerId);
*/
```

C:

```
typedef struct {

} Leaderboard;


Leaderboard* leaderboardCreate() {

}

void leaderboardAddScore(Leaderboard* obj, int playerId, int score) {

}

int leaderboardTop(Leaderboard* obj, int K) {

}

void leaderboardReset(Leaderboard* obj, int playerId) {

}

void leaderboardFree(Leaderboard* obj) {

}
```

```
/**
* Your Leaderboard struct will be instantiated and called as such:
* Leaderboard* obj = leaderboardCreate();
* leaderboardAddScore(obj, playerId, score);

* int param_2 = leaderboardTop(obj, K);

* leaderboardReset(obj, playerId);

* leaderboardFree(obj);
*/
```

Go:

```
type Leaderboard struct {

}


func Constructor() Leaderboard {

}


func (this *Leaderboard) AddScore(playerId int, score int) {

}


func (this *Leaderboard) Top(K int) int {

}


func (this *Leaderboard) Reset(playerId int) {

}


/**
* Your Leaderboard object will be instantiated and called as such:
* obj := Constructor();
```

```
* obj.AddScore(playerId,score);
* param_2 := obj.Top(K);
* obj.Reset(playerId);
*/
```

Kotlin:

```
class Leaderboard() {

fun addScore(playerId: Int, score: Int) {

}

fun top(K: Int): Int {

}

fun reset(playerId: Int) {

}

}

/**
* Your Leaderboard object will be instantiated and called as such:
* var obj = Leaderboard()
* obj.addScore(playerId,score)
* var param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

Swift:

```
class Leaderboard {

init() {

}

func addScore(_ playerId: Int, _ score: Int) {
```

```
}

func top(_ K: Int) -> Int {

}

func reset(_ playerId: Int) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* let obj = Leaderboard()
* obj.addScore(playerId, score)
* let ret_2: Int = obj.top(K)
* obj.reset(playerId)
*/
```

Rust:

```
struct Leaderboard {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Leaderboard {

fn new() -> Self {

}

fn add_score(&self, player_id: i32, score: i32) {

}

fn top(&self, k: i32) -> i32 {
```

```
}

fn reset(&self, player_id: i32) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* let obj = Leaderboard::new();
* obj.add_score(playerId, score);
* let ret_2: i32 = obj.top(K);
* obj.reset(playerId);
*/
```

Ruby:

```
class Leaderboard
def initialize()

end


=begin
:type player_id: Integer
:type score: Integer
:rtype: Void
=end
def add_score(player_id, score)

end


=begin
:type k: Integer
:rtype: Integer
=end
def top(k)

end
```

```
=begin
:type player_id: Integer
:rtype: Void
=end
def reset(player_id)

end



end

# Your Leaderboard object will be instantiated and called as such:
# obj = Leaderboard.new()
# obj.add_score(player_id, score)
# param_2 = obj.top(k)
# obj.reset(player_id)
```

PHP:

```
class Leaderboard {
/**
*/
function __construct() {

}

/**
* @param Integer $playerId
* @param Integer $score
* @return NULL
*/
function addScore($playerId, $score) {

}

/**
* @param Integer $K
* @return Integer
*/
function top($K) {

}
```

```
/**
* @param Integer $playerId
* @return NULL
*/
function reset($playerId) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* $obj = Leaderboard();
* $obj->addScore($playerId, $score);
* $ret_2 = $obj->top($K);
* $obj->reset($playerId);
*/
```

Dart:

```
class Leaderboard {

Leaderboard() {

}

void addScore(int playerId, int score) {

}

int top(int K) {

}

void reset(int playerId) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard obj = Leaderboard();
```

```
* obj.addScore(playerId,score);
* int param2 = obj.top(K);
* obj.reset(playerId);
*/
```

Scala:

```
class Leaderboard() {

def addScore(playerId: Int, score: Int): Unit = {

}

def top(K: Int): Int = {

}

def reset(playerId: Int): Unit = {

}

}

/**
* Your Leaderboard object will be instantiated and called as such:
* val obj = new Leaderboard()
* obj.addScore(playerId,score)
* val param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

Elixir:

```
defmodule Leaderboard do
@spec init_() :: any
def init_() do

end

@spec add_score(player_id :: integer, score :: integer) :: any
def add_score(player_id, score) do
```

```
end

@spec top(k :: integer) :: integer
def top(k) do

end

@spec reset(player_id :: integer) :: any
def reset(player_id) do

end
end

# Your functions will be called as such:
# Leaderboard.init_()
# Leaderboard.add_score(player_id, score)
# param_2 = Leaderboard.top(k)
# Leaderboard.reset(player_id)

# Leaderboard.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
-spec leaderboard_init_() -> any().
leaderboard_init_() ->
.

-spec leaderboard_add_score(PlayerId :: integer(), Score :: integer()) ->
any().
leaderboard_add_score(PlayerId, Score) ->
.

-spec leaderboard_top(K :: integer()) -> integer().
leaderboard_top(K) ->
.

-spec leaderboard_reset(PlayerId :: integer()) -> any().
leaderboard_reset(PlayerId) ->
.
```

```
%% Your functions will be called as such:
%% leaderboard_init_(),
%% leaderboard_add_score(PlayerId, Score),
%% Param_2 = leaderboard_top(K),
%% leaderboard_reset(PlayerId),

%% leaderboard_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket:

```
(define leaderboard%
(class object%
(super-new)

(init-field)

; add-score : exact-integer? exact-integer? -> void?
(define/public (add-score player-id score)
)
; top : exact-integer? -> exact-integer?
(define/public (top k)
)
; reset : exact-integer? -> void?
(define/public (reset player-id)
)))

;; Your leaderboard% object will be instantiated and called as such:
;; (define obj (new leaderboard%))
;; (send obj add-score player-id score)
;; (define param_2 (send obj top k))
;; (send obj reset player-id)
```

Solutions

C++ Solution:

```
/*
* Problem: Design A Leaderboard
* Difficulty: Medium
```

```
* Tags: hash, sort
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Leaderboard {
public:
Leaderboard() {

}

void addScore(int playerId, int score) {

}

int top(int K) {

}

void reset(int playerId) {

}
};

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard* obj = new Leaderboard();
* obj->addScore(playerId,score);
* int param_2 = obj->top(K);
* obj->reset(playerId);
*/
```

Java Solution:

```
/**
* Problem: Design A Leaderboard
* Difficulty: Medium
* Tags: hash, sort
*
```

```
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Leaderboard {

public Leaderboard() {

}

public void addScore(int playerId, int score) {

}

public int top(int K) {

}

public void reset(int playerId) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard obj = new Leaderboard();
* obj.addScore(playerId,score);
* int param_2 = obj.top(K);
* obj.reset(playerId);
*/
```

Python3 Solution:

```
"""
Problem: Design A Leaderboard
Difficulty: Medium
Tags: hash, sort

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
```

```
Space Complexity: O(n) for hash map
"""

class Leaderboard:

def __init__(self):


def addScore(self, playerId: int, score: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Leaderboard(object):

def __init__(self):


def addScore(self, playerId, score):
"""
:type playerId: int
:type score: int
:rtype: None
"""


def top(self, K):
"""
:type K: int
:rtype: int
"""


def reset(self, playerId):
"""
:type playerId: int
:rtype: None
"""
```

```
# Your Leaderboard object will be instantiated and called as such:
# obj = Leaderboard()
# obj.addScore(playerId,score)
# param_2 = obj.top(K)
# obj.reset(playerId)
```

JavaScript Solution:

```
/**
* Problem: Design A Leaderboard
* Difficulty: Medium
* Tags: hash, sort
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/


var Leaderboard = function() {

};

/**
* @param {number} playerId
* @param {number} score
* @return {void}
*/
Leaderboard.prototype.addScore = function(playerId, score) {

};

/**
* @param {number} K
* @return {number}
*/
Leaderboard.prototype.top = function(K) {

};
```

```
/**
* @param {number} playerId
* @return {void}
*/
Leaderboard.prototype.reset = function(playerId) {

};

/**
* Your Leaderboard object will be instantiated and called as such:
* var obj = new Leaderboard()
* obj.addScore(playerId,score)
* var param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

TypeScript Solution:

```
/**
* Problem: Design A Leaderboard
* Difficulty: Medium
* Tags: hash, sort
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Leaderboard {
constructor() {

}

addScore(playerId: number, score: number): void {

}

top(K: number): number {

}
```

```
reset(playerId: number): void {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* var obj = new Leaderboard()
* obj.addScore(playerId,score)
* var param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

C# Solution:

```
/*
* Problem: Design A Leaderboard
* Difficulty: Medium
* Tags: hash, sort
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class Leaderboard {

public Leaderboard() {

}

public void AddScore(int playerId, int score) {

}

public int Top(int K) {

}

public void Reset(int playerId) {
```

```
}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard obj = new Leaderboard();
* obj.AddScore(playerId,score);
* int param_2 = obj.Top(K);
* obj.Reset(playerId);
*/
```

C Solution:

```
/*
* Problem: Design A Leaderboard
* Difficulty: Medium
* Tags: hash, sort
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/




typedef struct {

} Leaderboard;


Leaderboard* leaderboardCreate() {

}

void leaderboardAddScore(Leaderboard* obj, int playerId, int score) {

}

int leaderboardTop(Leaderboard* obj, int K) {
```

```
}

void leaderboardReset(Leaderboard* obj, int playerId) {

}

void leaderboardFree(Leaderboard* obj) {

}

/**
* Your Leaderboard struct will be instantiated and called as such:
* Leaderboard* obj = leaderboardCreate();
* leaderboardAddScore(obj, playerId, score);

* int param_2 = leaderboardTop(obj, K);

* leaderboardReset(obj, playerId);

* leaderboardFree(obj);
*/
```

Go Solution:

```
// Problem: Design A Leaderboard
// Difficulty: Medium
// Tags: hash, sort
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

type Leaderboard struct {

}


func Constructor() Leaderboard {

}
```

```
func (this *Leaderboard) AddScore(playerId int, score int) {

}


func (this *Leaderboard) Top(K int) int {

}


func (this *Leaderboard) Reset(playerId int) {

}


/**
* Your Leaderboard object will be instantiated and called as such:
* obj := Constructor();
* obj.AddScore(playerId,score);
* param_2 := obj.Top(K);
* obj.Reset(playerId);
*/
```

Kotlin Solution:

```
class Leaderboard() {

fun addScore(playerId: Int, score: Int) {

}

fun top(K: Int): Int {

}

fun reset(playerId: Int) {

}

}
```

```
/**
* Your Leaderboard object will be instantiated and called as such:
* var obj = Leaderboard()
* obj.addScore(playerId,score)
* var param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

Swift Solution:

```
class Leaderboard {

init() {

}

func addScore(_ playerId: Int, _ score: Int) {

}

func top(_ K: Int) -> Int {

}

func reset(_ playerId: Int) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* let obj = Leaderboard()
* obj.addScore(playerId, score)
* let ret_2: Int = obj.top(K)
* obj.reset(playerId)
*/
```

Rust Solution:

```
// Problem: Design A Leaderboard
// Difficulty: Medium
// Tags: hash, sort
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

struct Leaderboard {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Leaderboard {

fn new() -> Self {

}

fn add_score(&self, player_id: i32, score: i32) {

}

fn top(&self, k: i32) -> i32 {

}

fn reset(&self, player_id: i32) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* let obj = Leaderboard::new();
* obj.add_score(playerId, score);
* let ret_2: i32 = obj.top(K);
* obj.reset(playerId);
*/
```

Ruby Solution:

```
class Leaderboard
def initialize()

end


=begin
:type player_id: Integer
:type score: Integer
:rtype: Void
=end
def add_score(player_id, score)

end


=begin
:type k: Integer
:rtype: Integer
=end
def top(k)

end


=begin
:type player_id: Integer
:rtype: Void
=end
def reset(player_id)

end


end

# Your Leaderboard object will be instantiated and called as such:
# obj = Leaderboard.new()
# obj.add_score(player_id, score)
# param_2 = obj.top(k)
```

```
# obj.reset(player_id)
```

PHP Solution:

```
class Leaderboard {
/**
*/
function __construct() {

}

/**
* @param Integer $playerId
* @param Integer $score
* @return NULL
*/
function addScore($playerId, $score) {

}

/**
* @param Integer $K
* @return Integer
*/
function top($K) {

}

/**
* @param Integer $playerId
* @return NULL
*/
function reset($playerId) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* $obj = Leaderboard();
* $obj->addScore($playerId, $score);
```

```
* $ret_2 = $obj->top($K);
* $obj->reset($playerId);
*/
```

Dart Solution:

```
class Leaderboard {

Leaderboard() {

}

void addScore(int playerId, int score) {

}

int top(int K) {

}

void reset(int playerId) {

}
}

/**
* Your Leaderboard object will be instantiated and called as such:
* Leaderboard obj = Leaderboard();
* obj.addScore(playerId,score);
* int param2 = obj.top(K);
* obj.reset(playerId);
*/
```

Scala Solution:

```
class Leaderboard() {

def addScore(playerId: Int, score: Int): Unit = {

}
```

```
def top(K: Int): Int = {

}

def reset(playerId: Int): Unit = {

}

}

/**
* Your Leaderboard object will be instantiated and called as such:
* val obj = new Leaderboard()
* obj.addScore(playerId,score)
* val param_2 = obj.top(K)
* obj.reset(playerId)
*/
```

Elixir Solution:

```
defmodule Leaderboard do
@spec init_() :: any
def init_() do

end

@spec add_score(player_id :: integer, score :: integer) :: any
def add_score(player_id, score) do

end

@spec top(k :: integer) :: integer
def top(k) do

end

@spec reset(player_id :: integer) :: any
def reset(player_id) do

end
end
```

```
# Your functions will be called as such:
# Leaderboard.init_()
# Leaderboard.add_score(player_id, score)
# param_2 = Leaderboard.top(k)
# Leaderboard.reset(player_id)

# Leaderboard.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec leaderboard_init_() -> any().
leaderboard_init_() ->
.

-spec leaderboard_add_score(PlayerId :: integer(), Score :: integer()) ->
any().
leaderboard_add_score(PlayerId, Score) ->
.

-spec leaderboard_top(K :: integer()) -> integer().
leaderboard_top(K) ->
.

-spec leaderboard_reset(PlayerId :: integer()) -> any().
leaderboard_reset(PlayerId) ->
.


%% Your functions will be called as such:
%% leaderboard_init_(),
%% leaderboard_add_score(PlayerId, Score),
%% Param_2 = leaderboard_top(K),
%% leaderboard_reset(PlayerId),

%% leaderboard_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket Solution:

```
(define leaderboard%
(class object%
(super-new)

(init-field)

; add-score : exact-integer? exact-integer? -> void?
(define/public (add-score player-id score)
)
; top : exact-integer? -> exact-integer?
(define/public (top k)
)
; reset : exact-integer? -> void?
(define/public (reset player-id)
)))

;; Your leaderboard% object will be instantiated and called as such:
;; (define obj (new leaderboard%))
;; (send obj add-score player-id score)
;; (define param_2 (send obj top k))
;; (send obj reset player-id)
```