Problem 2353: Design a Food Rating System

Problem Information

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

Problem Description

Design a food rating system that can do the following:

Modify

the rating of a food item listed in the system.

Return the highest-rated food item for a type of cuisine in the system.

Implement the

FoodRatings

class:

FoodRatings(String[] foods, String[] cuisines, int[] ratings)

Initializes the system. The food items are described by

foods

,

cuisines

and

ratings

, all of which have a length of

n

.

foods[i]

is the name of the

i

th

food,

cuisines[i]

is the type of cuisine of the

i

th

food, and

ratings[i]

is the initial rating of the

i

th

food.

void changeRating(String food, int newRating)

Changes the rating of the food item with the name

food

.

String highestRated(String cuisine)

Returns the name of the food item that has the highest rating for the given type of

cuisine

. If there is a tie, return the item with the

lexicographically smaller

name.

Note that a string

x

is lexicographically smaller than string

y

if

x

comes before

y

in dictionary order, that is, either

x

is a prefix of

y

, or if

i

is the first position such that

x[i] != y[i]

, then

x[i]

comes before

y[i]

in alphabetic order.

Example 1:

Input

["FoodRatings", "highestRated", "highestRated", "changeRating", "highestRated", "changeRating", "highestRated"] [[["kimchi", "miso", "sushi", "moussaka", "ramen", "bulgogi"], ["korean", "japanese", "japanese", "greek", "japanese", "korean"], [9, 12, 8, 15, 14, 7]], ["korean"], ["japanese"], ["sushi", 16], ["japanese"], ["ramen", 16], ["japanese"]]

Output

[null, "kimchi", "ramen", null, "sushi", null, "ramen"]

Explanation

FoodRatings foodRatings = new FoodRatings(["kimchi", "miso", "sushi", "moussaka", "ramen", "bulgogi"], ["korean", "japanese", "japanese", "greek", "japanese", "korean"], [9, 12, 8, 15, 14,

7]); foodRatings.highestRated("korean"); // return "kimchi" // "kimchi" is the highest rated korean food with a rating of 9. foodRatings.highestRated("japanese"); // return "ramen" // "ramen" is the highest rated japanese food with a rating of 14. foodRatings.changeRating("sushi", 16); // "sushi" now has a rating of 16. foodRatings.highestRated("japanese"); // return "sushi" // "sushi" is the highest rated japanese food with a rating of 16. foodRatings.changeRating("ramen", 16); // "ramen" now has a rating of 16. foodRatings.highestRated("japanese"); // return "ramen" // Both "sushi" and "ramen" have a rating of 16. // However, "ramen" is lexicographically smaller than "sushi".

Constraints:

1 <= n <= 2 * 10

4

n == foods.length == cuisines.length == ratings.length

1 <= foods[i].length, cuisines[i].length <= 10

foods[i]

,

cuisines[i]

consist of lowercase English letters.

1 <= ratings[i] <= 10

8

All the strings in

foods

are

distinct

.

food

will be the name of a food item in the system across all calls to

changeRating

.

cuisine

will be a type of cuisine of

at least one

food item in the system across all calls to

highestRated

.

At most

2 * 10

4

calls

in total

will be made to

changeRating

and

highestRated

.

Code Snippets

C++:

```
class FoodRatings {
public:
FoodRatings(vector<string>& foods, vector<string>& cuisines, vector<int>&
ratings) {

}

void changeRating(string food, int newRating) {

}

string highestRated(string cuisine) {

}
};

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings* obj = new FoodRatings(foods, cuisines, ratings);
* obj->changeRating(food,newRating);
* string param_2 = obj->highestRated(cuisine);
*/
```

Java:

```
class FoodRatings {

public FoodRatings(String[] foods, String[] cuisines, int[] ratings) {

}

public void changeRating(String food, int newRating) {

}
```

```
public String highestRated(String cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings obj = new FoodRatings(foods, cuisines, ratings);
* obj.changeRating(food,newRating);
* String param_2 = obj.highestRated(cuisine);
*/
```

Python3:

```
class FoodRatings:

def __init__(self, foods: List[str], cuisines: List[str], ratings:
List[int]):


def changeRating(self, food: str, newRating: int) -> None:


def highestRated(self, cuisine: str) -> str:



# Your FoodRatings object will be instantiated and called as such:
# obj = FoodRatings(foods, cuisines, ratings)
# obj.changeRating(food,newRating)
# param_2 = obj.highestRated(cuisine)
```

Python:

```
class FoodRatings(object):

def __init__(self, foods, cuisines, ratings):
"""
:type foods: List[str]
:type cuisines: List[str]
:type ratings: List[int]
"""
```

```
def changeRating(self, food, newRating):
"""
:type food: str
:type newRating: int
:rtype: None
"""


def highestRated(self, cuisine):
"""
:type cuisine: str
:rtype: str
"""



# Your FoodRatings object will be instantiated and called as such:
# obj = FoodRatings(foods, cuisines, ratings)
# obj.changeRating(food,newRating)
# param_2 = obj.highestRated(cuisine)
```

JavaScript:

```
/**
* @param {string[]} foods
* @param {string[]} cuisines
* @param {number[]} ratings
*/
var FoodRatings = function(foods, cuisines, ratings) {

};

/**
* @param {string} food
* @param {number} newRating
* @return {void}
*/
FoodRatings.prototype.changeRating = function(food, newRating) {

};
```

```
/**
* @param {string} cuisine
* @return {string}
*/
FoodRatings.prototype.highestRated = function(cuisine) {

};

/**
* Your FoodRatings object will be instantiated and called as such:
* var obj = new FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* var param_2 = obj.highestRated(cuisine)
*/
```

TypeScript:

```
class FoodRatings {
constructor(foods: string[], cuisines: string[], ratings: number[]) {

}

changeRating(food: string, newRating: number): void {

}

highestRated(cuisine: string): string {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* var obj = new FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* var param_2 = obj.highestRated(cuisine)
*/
```

C#:

```
public class FoodRatings {

public FoodRatings(string[] foods, string[] cuisines, int[] ratings) {

}

public void ChangeRating(string food, int newRating) {

}

public string HighestRated(string cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings obj = new FoodRatings(foods, cuisines, ratings);
* obj.ChangeRating(food,newRating);
* string param_2 = obj.HighestRated(cuisine);
*/
```

C:

```
typedef struct {

} FoodRatings;


FoodRatings* foodRatingsCreate(char** foods, int foodsSize, char** cuisines,
int cuisinesSize, int* ratings, int ratingsSize) {

}

void foodRatingsChangeRating(FoodRatings* obj, char* food, int newRating) {

}

char* foodRatingsHighestRated(FoodRatings* obj, char* cuisine) {
```

```
}

void foodRatingsFree(FoodRatings* obj) {

}

/**
* Your FoodRatings struct will be instantiated and called as such:
* FoodRatings* obj = foodRatingsCreate(foods, foodsSize, cuisines,
cuisinesSize, ratings, ratingsSize);
* foodRatingsChangeRating(obj, food, newRating);

* char* param_2 = foodRatingsHighestRated(obj, cuisine);

* foodRatingsFree(obj);
*/
```

Go:

```
type FoodRatings struct {

}


func Constructor(foods []string, cuisines []string, ratings []int)
FoodRatings {

}


func (this *FoodRatings) ChangeRating(food string, newRating int) {

}


func (this *FoodRatings) HighestRated(cuisine string) string {

}


/**
* Your FoodRatings object will be instantiated and called as such:
```

```
* obj := Constructor(foods, cuisines, ratings);
* obj.ChangeRating(food,newRating);
* param_2 := obj.HighestRated(cuisine);
*/
```

Kotlin:

```
class FoodRatings(foods: Array<String>, cuisines: Array<String>, ratings:
IntArray) {

fun changeRating(food: String, newRating: Int) {

}

fun highestRated(cuisine: String): String {

}

}

/**
* Your FoodRatings object will be instantiated and called as such:
* var obj = FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* var param_2 = obj.highestRated(cuisine)
*/
```

Swift:

```
class FoodRatings {

init(_ foods: [String], _ cuisines: [String], _ ratings: [Int]) {

}

func changeRating(_ food: String, _ newRating: Int) {

}

func highestRated(_ cuisine: String) -> String {
```

```
}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* let obj = FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food, newRating)
* let ret_2: String = obj.highestRated(cuisine)
*/
```

Rust:

```
struct FoodRatings {

}


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

fn new(foods: Vec<String>, cuisines: Vec<String>, ratings: Vec<i32>) -> Self
{

}

fn change_rating(&self, food: String, new_rating: i32) {

}

fn highest_rated(&self, cuisine: String) -> String {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* let obj = FoodRatings::new(foods, cuisines, ratings);
* obj.change_rating(food, newRating);
* let ret_2: String = obj.highest_rated(cuisine);
```

```
*/
```

Ruby:

```
class FoodRatings

=begin
:type foods: String[]
:type cuisines: String[]
:type ratings: Integer[]
=end
def initialize(foods, cuisines, ratings)

end


=begin
:type food: String
:type new_rating: Integer
:rtype: Void
=end
def change_rating(food, new_rating)

end


=begin
:type cuisine: String
:rtype: String
=end
def highest_rated(cuisine)

end


end

# Your FoodRatings object will be instantiated and called as such:
# obj = FoodRatings.new(foods, cuisines, ratings)
# obj.change_rating(food, new_rating)
# param_2 = obj.highest_rated(cuisine)
```

PHP:

```
class FoodRatings {
/**
* @param String[] $foods
* @param String[] $cuisines
* @param Integer[] $ratings
*/
function __construct($foods, $cuisines, $ratings) {

}

/**
* @param String $food
* @param Integer $newRating
* @return NULL
*/
function changeRating($food, $newRating) {

}

/**
* @param String $cuisine
* @return String
*/
function highestRated($cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* $obj = FoodRatings($foods, $cuisines, $ratings);
* $obj->changeRating($food, $newRating);
* $ret_2 = $obj->highestRated($cuisine);
*/
```

Dart:

```
class FoodRatings {

FoodRatings(List<String> foods, List<String> cuisines, List<int> ratings) {
```

```
}

void changeRating(String food, int newRating) {

}

String highestRated(String cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings obj = FoodRatings(foods, cuisines, ratings);
* obj.changeRating(food,newRating);
* String param2 = obj.highestRated(cuisine);
*/
```

Scala:

```
class FoodRatings(_foods: Array[String], _cuisines: Array[String], _ratings:
Array[Int]) {

def changeRating(food: String, newRating: Int): Unit = {

}

def highestRated(cuisine: String): String = {

}

}

/**
* Your FoodRatings object will be instantiated and called as such:
* val obj = new FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* val param_2 = obj.highestRated(cuisine)
*/
```

Elixir:

```
defmodule FoodRatings do
@spec init_(foods :: [String.t], cuisines :: [String.t], ratings ::
[integer]) :: any
def init_(foods, cuisines, ratings) do

end

@spec change_rating(food :: String.t, new_rating :: integer) :: any
def change_rating(food, new_rating) do

end

@spec highest_rated(cuisine :: String.t) :: String.t
def highest_rated(cuisine) do

end
end

# Your functions will be called as such:
# FoodRatings.init_(foods, cuisines, ratings)
# FoodRatings.change_rating(food, new_rating)
# param_2 = FoodRatings.highest_rated(cuisine)

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

Erlang:

```
-spec food_ratings_init_(Foods :: [unicode:unicode_binary()], Cuisines ::
[unicode:unicode_binary()], Ratings :: [integer()]) -> any().
food_ratings_init_(Foods, Cuisines, Ratings) ->
.

-spec food_ratings_change_rating(Food :: unicode:unicode_binary(), NewRating
:: integer()) -> any().
food_ratings_change_rating(Food, NewRating) ->
.

-spec food_ratings_highest_rated(Cuisine :: unicode:unicode_binary()) ->
unicode:unicode_binary().
food_ratings_highest_rated(Cuisine) ->
```

```
.


%% Your functions will be called as such:
%% food_ratings_init_(Foods, Cuisines, Ratings),
%% food_ratings_change_rating(Food, NewRating),
%% Param_2 = food_ratings_highest_rated(Cuisine),

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

Racket:

```
(define food-ratings%
(class object%
(super-new)

; foods : (listof string?)
; cuisines : (listof string?)
; ratings : (listof exact-integer?)
(init-field
foods
cuisines
ratings)

; change-rating : string? exact-integer? -> void?
(define/public (change-rating food new-rating)
)
; highest-rated : string? -> string?
(define/public (highest-rated cuisine)
)))

;; Your food-ratings% object will be instantiated and called as such:
;; (define obj (new food-ratings% [foods foods] [cuisines cuisines] [ratings
ratings]))
;; (send obj change-rating food new-rating)
;; (define param_2 (send obj highest-rated cuisine))
```

Solutions

C++ Solution:

```
/*
* Problem: Design a Food Rating System
* Difficulty: Medium
* Tags: array, string, graph, hash, queue, heap
*
* 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 FoodRatings {
public:
FoodRatings(vector<string>& foods, vector<string>& cuisines, vector<int>&
ratings) {

}

void changeRating(string food, int newRating) {

}

string highestRated(string cuisine) {

}
};

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings* obj = new FoodRatings(foods, cuisines, ratings);
* obj->changeRating(food,newRating);
* string param_2 = obj->highestRated(cuisine);
*/
```

Java Solution:

```
/**
* Problem: Design a Food Rating System
* Difficulty: Medium
* Tags: array, string, graph, hash, queue, heap
*
* 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 FoodRatings {

public FoodRatings(String[] foods, String[] cuisines, int[] ratings) {

}

public void changeRating(String food, int newRating) {

}

public String highestRated(String cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings obj = new FoodRatings(foods, cuisines, ratings);
* obj.changeRating(food,newRating);
* String param_2 = obj.highestRated(cuisine);
*/
```

Python3 Solution:

```
"""
Problem: Design a Food Rating System
Difficulty: Medium
Tags: array, string, graph, hash, queue, heap

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 FoodRatings:

def __init__(self, foods: List[str], cuisines: List[str], ratings:
```

```
List[int]):


def changeRating(self, food: str, newRating: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class FoodRatings(object):

def __init__(self, foods, cuisines, ratings):
"""
:type foods: List[str]
:type cuisines: List[str]
:type ratings: List[int]
"""


def changeRating(self, food, newRating):
"""
:type food: str
:type newRating: int
:rtype: None
"""


def highestRated(self, cuisine):
"""
:type cuisine: str
:rtype: str
"""




# Your FoodRatings object will be instantiated and called as such:
# obj = FoodRatings(foods, cuisines, ratings)
# obj.changeRating(food,newRating)
# param_2 = obj.highestRated(cuisine)
```

JavaScript Solution:

```
/**
* Problem: Design a Food Rating System
* Difficulty: Medium
* Tags: array, string, graph, hash, queue, heap
*
* 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 {string[]} foods
* @param {string[]} cuisines
* @param {number[]} ratings
*/
var FoodRatings = function(foods, cuisines, ratings) {

};

/**
* @param {string} food
* @param {number} newRating
* @return {void}
*/
FoodRatings.prototype.changeRating = function(food, newRating) {

};

/**
* @param {string} cuisine
* @return {string}
*/
FoodRatings.prototype.highestRated = function(cuisine) {

};

/**
* Your FoodRatings object will be instantiated and called as such:
* var obj = new FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* var param_2 = obj.highestRated(cuisine)
*/
```

TypeScript Solution:

```
/**
* Problem: Design a Food Rating System
* Difficulty: Medium
* Tags: array, string, graph, hash, queue, heap
*
* 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 FoodRatings {
constructor(foods: string[], cuisines: string[], ratings: number[]) {

}

changeRating(food: string, newRating: number): void {

}

highestRated(cuisine: string): string {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* var obj = new FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* var param_2 = obj.highestRated(cuisine)
*/
```

C# Solution:

```
/*
* Problem: Design a Food Rating System
* Difficulty: Medium
* Tags: array, string, graph, hash, queue, heap
*
* 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 FoodRatings {

public FoodRatings(string[] foods, string[] cuisines, int[] ratings) {

}

public void ChangeRating(string food, int newRating) {

}

public string HighestRated(string cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings obj = new FoodRatings(foods, cuisines, ratings);
* obj.ChangeRating(food,newRating);
* string param_2 = obj.HighestRated(cuisine);
*/
```

C Solution:

```
/*
* Problem: Design a Food Rating System
* Difficulty: Medium
* Tags: array, string, graph, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {
```

```
} FoodRatings;


FoodRatings* foodRatingsCreate(char** foods, int foodsSize, char** cuisines,
int cuisinesSize, int* ratings, int ratingsSize) {

}

void foodRatingsChangeRating(FoodRatings* obj, char* food, int newRating) {

}

char* foodRatingsHighestRated(FoodRatings* obj, char* cuisine) {

}

void foodRatingsFree(FoodRatings* obj) {

}

/**
* Your FoodRatings struct will be instantiated and called as such:
* FoodRatings* obj = foodRatingsCreate(foods, foodsSize, cuisines,
cuisinesSize, ratings, ratingsSize);
* foodRatingsChangeRating(obj, food, newRating);

* char* param_2 = foodRatingsHighestRated(obj, cuisine);

* foodRatingsFree(obj);
*/
```

Go Solution:

```
// Problem: Design a Food Rating System
// Difficulty: Medium
// Tags: array, string, graph, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
type FoodRatings struct {

}


func Constructor(foods []string, cuisines []string, ratings []int)
FoodRatings {

}


func (this *FoodRatings) ChangeRating(food string, newRating int) {

}


func (this *FoodRatings) HighestRated(cuisine string) string {

}


/**
* Your FoodRatings object will be instantiated and called as such:
* obj := Constructor(foods, cuisines, ratings);
* obj.ChangeRating(food,newRating);
* param_2 := obj.HighestRated(cuisine);
*/
```

Kotlin Solution:

```
class FoodRatings(foods: Array<String>, cuisines: Array<String>, ratings:
IntArray) {

fun changeRating(food: String, newRating: Int) {

}

fun highestRated(cuisine: String): String {

}
```

```
}

/**
* Your FoodRatings object will be instantiated and called as such:
* var obj = FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* var param_2 = obj.highestRated(cuisine)
*/
```

Swift Solution:

```
class FoodRatings {

init(_ foods: [String], _ cuisines: [String], _ ratings: [Int]) {

}

func changeRating(_ food: String, _ newRating: Int) {

}

func highestRated(_ cuisine: String) -> String {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* let obj = FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food, newRating)
* let ret_2: String = obj.highestRated(cuisine)
*/
```

Rust Solution:

```
// Problem: Design a Food Rating System
// Difficulty: Medium
// Tags: array, string, graph, hash, queue, heap
//
```

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

struct FoodRatings {

}


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

fn new(foods: Vec<String>, cuisines: Vec<String>, ratings: Vec<i32>) -> Self
{

}

fn change_rating(&self, food: String, new_rating: i32) {

}

fn highest_rated(&self, cuisine: String) -> String {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* let obj = FoodRatings::new(foods, cuisines, ratings);
* obj.change_rating(food, newRating);
* let ret_2: String = obj.highest_rated(cuisine);
*/
```

Ruby Solution:

```
class FoodRatings

=begin
```

```
:type foods: String[]
:type cuisines: String[]
:type ratings: Integer[]
=end
def initialize(foods, cuisines, ratings)

end


=begin
:type food: String
:type new_rating: Integer
:rtype: Void
=end
def change_rating(food, new_rating)

end


=begin
:type cuisine: String
:rtype: String
=end
def highest_rated(cuisine)

end


end

# Your FoodRatings object will be instantiated and called as such:
# obj = FoodRatings.new(foods, cuisines, ratings)
# obj.change_rating(food, new_rating)
# param_2 = obj.highest_rated(cuisine)
```

PHP Solution:

```
class FoodRatings {
/**
* @param String[] $foods
* @param String[] $cuisines
```

```
* @param Integer[] $ratings
*/
function __construct($foods, $cuisines, $ratings) {

}

/**
* @param String $food
* @param Integer $newRating
* @return NULL
*/
function changeRating($food, $newRating) {

}

/**
* @param String $cuisine
* @return String
*/
function highestRated($cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* $obj = FoodRatings($foods, $cuisines, $ratings);
* $obj->changeRating($food, $newRating);
* $ret_2 = $obj->highestRated($cuisine);
*/
```

Dart Solution:

```
class FoodRatings {

FoodRatings(List<String> foods, List<String> cuisines, List<int> ratings) {

}

void changeRating(String food, int newRating) {
```

```
}

String highestRated(String cuisine) {

}
}

/**
* Your FoodRatings object will be instantiated and called as such:
* FoodRatings obj = FoodRatings(foods, cuisines, ratings);
* obj.changeRating(food,newRating);
* String param2 = obj.highestRated(cuisine);
*/
```

Scala Solution:

```
class FoodRatings(_foods: Array[String], _cuisines: Array[String], _ratings:
Array[Int]) {

def changeRating(food: String, newRating: Int): Unit = {

}

def highestRated(cuisine: String): String = {

}

}

/**
* Your FoodRatings object will be instantiated and called as such:
* val obj = new FoodRatings(foods, cuisines, ratings)
* obj.changeRating(food,newRating)
* val param_2 = obj.highestRated(cuisine)
*/
```

Elixir Solution:

```
defmodule FoodRatings do
@spec init_(foods :: [String.t], cuisines :: [String.t], ratings ::
[integer]) :: any
```

```
def init_(foods, cuisines, ratings) do

end

@spec change_rating(food :: String.t, new_rating :: integer) :: any
def change_rating(food, new_rating) do

end

@spec highest_rated(cuisine :: String.t) :: String.t
def highest_rated(cuisine) do

end
end

# Your functions will be called as such:
# FoodRatings.init_(foods, cuisines, ratings)
# FoodRatings.change_rating(food, new_rating)
# param_2 = FoodRatings.highest_rated(cuisine)

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

Erlang Solution:

```
-spec food_ratings_init_(Foods :: [unicode:unicode_binary()], Cuisines ::
[unicode:unicode_binary()], Ratings :: [integer()]) -> any().
food_ratings_init_(Foods, Cuisines, Ratings) ->
.

-spec food_ratings_change_rating(Food :: unicode:unicode_binary(), NewRating
:: integer()) -> any().
food_ratings_change_rating(Food, NewRating) ->
.

-spec food_ratings_highest_rated(Cuisine :: unicode:unicode_binary()) ->
unicode:unicode_binary().
food_ratings_highest_rated(Cuisine) ->
.
```

```
%% Your functions will be called as such:
%% food_ratings_init_(Foods, Cuisines, Ratings),
%% food_ratings_change_rating(Food, NewRating),
%% Param_2 = food_ratings_highest_rated(Cuisine),

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

Racket Solution:

```
(define food-ratings%
(class object%
(super-new)

; foods : (listof string?)
; cuisines : (listof string?)
; ratings : (listof exact-integer?)
(init-field
foods
cuisines
ratings)

; change-rating : string? exact-integer? -> void?
(define/public (change-rating food new-rating)
)
; highest-rated : string? -> string?
(define/public (highest-rated cuisine)
)))

;; Your food-ratings% object will be instantiated and called as such:
;; (define obj (new food-ratings% [foods foods] [cuisines cuisines] [ratings
ratings]))
;; (send obj change-rating food new-rating)
;; (define param_2 (send obj highest-rated cuisine))
```