

# Problem 478: Generate Random Point in a Circle

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Given the radius and the position of the center of a circle, implement the function

`randPoint`

which generates a uniform random point inside the circle.

Implement the

`Solution`

class:

`Solution(double radius, double x_center, double y_center)`

initializes the object with the radius of the circle

`radius`

and the position of the center

`(x_center, y_center)`

.

`randPoint()`

returns a random point inside the circle. A point on the circumference of the circle is considered to be in the circle. The answer is returned as an array

[x, y]

.

Example 1:

Input

["Solution", "randPoint", "randPoint", "randPoint"] [[1.0, 0.0, 0.0], [], [], []]

Output

[null, [-0.02493, -0.38077], [0.82314, 0.38945], [0.36572, 0.17248]]

Explanation

Solution solution = new Solution(1.0, 0.0, 0.0); solution.randPoint(); // return [-0.02493, -0.38077] solution.randPoint(); // return [0.82314, 0.38945] solution.randPoint(); // return [0.36572, 0.17248]

Constraints:

$0 < \text{radius} \leq 10$

8

-10

7

$\leq x\_center, y\_center \leq 10$

7

At most

3 \* 10

4

calls will be made to

randPoint

.

## Code Snippets

### C++:

```
class Solution {
public:
    Solution(double radius, double x_center, double y_center) {

    }

    vector<double> randPoint() {

    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(radius, x_center, y_center);
 * vector<double> param_1 = obj->randPoint();
 */
```

### Java:

```
class Solution {

    public Solution(double radius, double x_center, double y_center) {

    }

    public double[] randPoint() {
```

```

}
}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(radius, x_center, y_center);
 * double[] param_1 = obj.randPoint();
 */

```

### Python3:

```

class Solution:

    def __init__(self, radius: float, x_center: float, y_center: float):

    def randPoint(self) -> List[float]:

    # Your Solution object will be instantiated and called as such:
    # obj = Solution(radius, x_center, y_center)
    # param_1 = obj.randPoint()

```

### Python:

```

class Solution(object):

    def __init__(self, radius, x_center, y_center):
        """
        :type radius: float
        :type x_center: float
        :type y_center: float
        """

    def randPoint(self):
        """
        :rtype: List[float]
        """

```

```
# Your Solution object will be instantiated and called as such:
# obj = Solution(radius, x_center, y_center)
# param_1 = obj.randPoint()
```

### JavaScript:

```
/**
 * @param {number} radius
 * @param {number} x_center
 * @param {number} y_center
 */
var Solution = function(radius, x_center, y_center) {

};

/**
 * @return {number[]}
 */
Solution.prototype.randPoint = function() {

};

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = new Solution(radius, x_center, y_center)
 * var param_1 = obj.randPoint()
 */
```

### TypeScript:

```
class Solution {
  constructor(radius: number, x_center: number, y_center: number) {

  }

  randPoint(): number[] {

  }
}
```

```

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = new Solution(radius, x_center, y_center)
 * var param_1 = obj.randPoint()
 */

```

## C#:

```

public class Solution {

    public Solution(double radius, double x_center, double y_center) {

    }

    public double[] RandPoint() {

    }

}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(radius, x_center, y_center);
 * double[] param_1 = obj.RandPoint();
 */

```

## C:

```

typedef struct {

} Solution;

Solution* solutionCreate(double radius, double x_center, double y_center) {

}

double* solutionRandPoint(Solution* obj, int* retSize) {

}

```

```

void solutionFree(Solution* obj) {

}

/**
 * Your Solution struct will be instantiated and called as such:
 * Solution* obj = solutionCreate(radius, x_center, y_center);
 * double* param_1 = solutionRandPoint(obj, retSize);

 * solutionFree(obj);
 */

```

## Go:

```

type Solution struct {

}

func Constructor(radius float64, x_center float64, y_center float64) Solution
{

}

func (this *Solution) RandPoint() []float64 {

}

/**
 * Your Solution object will be instantiated and called as such:
 * obj := Constructor(radius, x_center, y_center);
 * param_1 := obj.RandPoint();
 */

```

## Kotlin:

```

class Solution(radius: Double, x_center: Double, y_center: Double) {

fun randPoint(): DoubleArray {

```

```

}

}

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = Solution(radius, x_center, y_center)
 * var param_1 = obj.randPoint()
 */

```

### Swift:

```

class Solution {

    init(_ radius: Double, _ x_center: Double, _ y_center: Double) {

    }

    func randPoint() -> [Double] {

    }

}

/**
 * Your Solution object will be instantiated and called as such:
 * let obj = Solution(radius, x_center, y_center)
 * let ret_1: [Double] = obj.randPoint()
 */

```

### Rust:

```

struct Solution {

}

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

```

```

impl Solution {

  fn new(radius: f64, x_center: f64, y_center: f64) -> Self {

  }

  fn rand_point(&self) -> Vec<f64> {

  }
}

/**
 * Your Solution object will be instantiated and called as such:
 * let obj = Solution::new(radius, x_center, y_center);
 * let ret_1: Vec<f64> = obj.rand_point();
 */

```

## Ruby:

```

class Solution

  =begin
  :type radius: Float
  :type x_center: Float
  :type y_center: Float
  =end
  def initialize(radius, x_center, y_center)

  end

  =begin
  :rtype: Float[]
  =end
  def rand_point()

  end

end

# Your Solution object will be instantiated and called as such:

```

```
# obj = Solution.new(radius, x_center, y_center)
# param_1 = obj.rand_point()
```

## PHP:

```
class Solution {
    /**
     * @param Float $radius
     * @param Float $x_center
     * @param Float $y_center
     */
    function __construct($radius, $x_center, $y_center) {

    }

    /**
     * @return Float[]
     */
    function randPoint() {

    }
}

/**
 * Your Solution object will be instantiated and called as such:
 * $obj = Solution($radius, $x_center, $y_center);
 * $ret_1 = $obj->randPoint();
 */
```

## Dart:

```
class Solution {

    Solution(double radius, double x_center, double y_center) {

    }

    List<double> randPoint() {

    }
}
```

```

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = Solution(radius, x_center, y_center);
 * List<double> param1 = obj.randPoint();
 */

```

## Scala:

```

class Solution(_radius: Double, _x_center: Double, _y_center: Double) {

  def randPoint(): Array[Double] = {

  }

}

/**
 * Your Solution object will be instantiated and called as such:
 * val obj = new Solution(radius, x_center, y_center)
 * val param_1 = obj.randPoint()
 */

```

## Elixir:

```

defmodule Solution do
  @spec init_(radius :: float, x_center :: float, y_center :: float) :: any
  def init_(radius, x_center, y_center) do

  end

  @spec rand_point() :: [float]
  def rand_point() do

  end

  end

  # Your functions will be called as such:
  # Solution.init_(radius, x_center, y_center)
  # param_1 = Solution.rand_point()

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

```

## Erlang:

```
-spec solution_init_(Radius :: float(), X_center :: float(), Y_center ::
float()) -> any().
solution_init_(Radius, X_center, Y_center) ->
.

-spec solution_rand_point() -> [float()].
solution_rand_point() ->
.

%% Your functions will be called as such:
%% solution_init_(Radius, X_center, Y_center),
%% Param_1 = solution_rand_point(),

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

## Racket:

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

; radius : flonum?
; x_center : flonum?
; y_center : flonum?
(init-field
radius
x_center
y_center)

; rand-point : -> (listof flonum?)
(define/public (rand-point)
)))

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [radius radius] [x_center x_center] [y_center
y_center]))
;; (define param_1 (send obj rand-point))
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Generate Random Point in a Circle
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public:
    Solution(double radius, double x_center, double y_center) {

    }

    vector<double> randPoint() {

    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(radius, x_center, y_center);
 * vector<double> param_1 = obj->randPoint();
 */
```

### Java Solution:

```
/**
 * Problem: Generate Random Point in a Circle
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
```

```

*/

class Solution {

public Solution(double radius, double x_center, double y_center) {

}

public double[] randPoint() {

}

}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(radius, x_center, y_center);
 * double[] param_1 = obj.randPoint();
 */

```

### Python3 Solution:

```

"""
Problem: Generate Random Point in a Circle
Difficulty: Medium
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:

    def __init__(self, radius: float, x_center: float, y_center: float):

    def randPoint(self) -> List[float]:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):

    def __init__(self, radius, x_center, y_center):
        """
        :type radius: float
        :type x_center: float
        :type y_center: float
        """

    def randPoint(self):
        """
        :rtype: List[float]
        """

# Your Solution object will be instantiated and called as such:
# obj = Solution(radius, x_center, y_center)
# param_1 = obj.randPoint()

```

## JavaScript Solution:

```

/**
 * Problem: Generate Random Point in a Circle
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * @param {number} radius
 * @param {number} x_center
 * @param {number} y_center
 */
var Solution = function(radius, x_center, y_center) {

};

```

```

/**
 * @return {number[]}
 */
Solution.prototype.randPoint = function() {

};

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = new Solution(radius, x_center, y_center)
 * var param_1 = obj.randPoint()
 */

```

### TypeScript Solution:

```

/**
 * Problem: Generate Random Point in a Circle
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
  constructor(radius: number, x_center: number, y_center: number) {

  }

  randPoint(): number[] {

  }
}

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = new Solution(radius, x_center, y_center)
 * var param_1 = obj.randPoint()
 */

```

## C# Solution:

```
/*
 * Problem: Generate Random Point in a Circle
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

public class Solution {

    public Solution(double radius, double x_center, double y_center) {

    }

    public double[] RandPoint() {

    }

}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(radius, x_center, y_center);
 * double[] param_1 = obj.RandPoint();
 */
```

## C Solution:

```
/*
 * Problem: Generate Random Point in a Circle
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */
```

```

typedef struct {

} Solution;

Solution* solutionCreate(double radius, double x_center, double y_center) {

}

double* solutionRandPoint(Solution* obj, int* retSize) {

}

void solutionFree(Solution* obj) {

}

/**
 * Your Solution struct will be instantiated and called as such:
 * Solution* obj = solutionCreate(radius, x_center, y_center);
 * double* param_1 = solutionRandPoint(obj, retSize);
 *
 * solutionFree(obj);
 */

```

## Go Solution:

```

// Problem: Generate Random Point in a Circle
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

type Solution struct {

}

```

```

func Constructor(radius float64, x_center float64, y_center float64) Solution
{

}

func (this *Solution) RandPoint() []float64 {

}

/**
 * Your Solution object will be instantiated and called as such:
 * obj := Constructor(radius, x_center, y_center);
 * param_1 := obj.RandPoint();
 */

```

### Kotlin Solution:

```

class Solution(radius: Double, x_center: Double, y_center: Double) {

    fun randPoint(): DoubleArray {

    }

}

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = Solution(radius, x_center, y_center)
 * var param_1 = obj.randPoint()
 */

```

### Swift Solution:

```

class Solution {

    init(_ radius: Double, _ x_center: Double, _ y_center: Double) {

    }

}

```

```

func randPoint() -> [Double] {

}

}

/**
 * Your Solution object will be instantiated and called as such:
 * let obj = Solution(radius, x_center, y_center)
 * let ret_1: [Double] = obj.randPoint()
 */

```

## Rust Solution:

```

// Problem: Generate Random Point in a Circle
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

struct Solution {

}

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

    fn new(radius: f64, x_center: f64, y_center: f64) -> Self {

    }

    fn rand_point(&self) -> Vec<f64> {

    }

}

```

```

/**
 * Your Solution object will be instantiated and called as such:
 * let obj = Solution::new(radius, x_center, y_center);
 * let ret_1: Vec<f64> = obj.rand_point();
 */

```

## Ruby Solution:

```

class Solution

  =begin
  :type radius: Float
  :type x_center: Float
  :type y_center: Float
  =end

  def initialize(radius, x_center, y_center)

  end

  =begin
  :rtype: Float[]
  =end

  def rand_point()

  end

end

# Your Solution object will be instantiated and called as such:
# obj = Solution.new(radius, x_center, y_center)
# param_1 = obj.rand_point()

```

## PHP Solution:

```

class Solution {
    /**
     * @param Float $radius
     * @param Float $x_center
     */

```

```

* @param Float $y_center
*/
function __construct($radius, $x_center, $y_center) {

}

/**
* @return Float[]
*/
function randPoint() {

}

}

/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($radius, $x_center, $y_center);
* $ret_1 = $obj->randPoint();
*/

```

### Dart Solution:

```

class Solution {

  Solution(double radius, double x_center, double y_center) {

  }

  List<double> randPoint() {

  }

}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = Solution(radius, x_center, y_center);
* List<double> param1 = obj.randPoint();
*/

```

### Scala Solution:

```

class Solution(_radius: Double, _x_center: Double, _y_center: Double) {

  def randPoint(): Array[Double] = {

  }

}

/**
 * Your Solution object will be instantiated and called as such:
 * val obj = new Solution(radius, x_center, y_center)
 * val param_1 = obj.randPoint()
 */

```

### Elixir Solution:

```

defmodule Solution do
  @spec init_(radius :: float, x_center :: float, y_center :: float) :: any
  def init_(radius, x_center, y_center) do

  end

  @spec rand_point() :: [float]
  def rand_point() do

  end
end

# Your functions will be called as such:
# Solution.init_(radius, x_center, y_center)
# param_1 = Solution.rand_point()

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

```

### Erlang Solution:

```

-spec solution_init_(Radius :: float(), X_center :: float(), Y_center ::
float()) -> any().
solution_init_(Radius, X_center, Y_center) ->
.

```

```

-spec solution_rand_point() -> [float()].
solution_rand_point() ->
.

%% Your functions will be called as such:
%% solution_init_(Radius, X_center, Y_center),
%% Param_1 = solution_rand_point(),

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

```

### Racket Solution:

```

(define solution%
  (class object%
    (super-new)

    ; radius : flonum?
    ; x_center : flonum?
    ; y_center : flonum?
    (init-field
      radius
      x_center
      y_center)

    ; rand-point : -> (listof flonum?)
    (define/public (rand-point)
      )))

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [radius radius] [x_center x_center] [y_center
y_center]))
;; (define param_1 (send obj rand-point))

```