# 1. Python: Multiset Implementation

A multiset is the same as a set except that an element might occur more than once in a multiset. Implement a multiset data structure in Python. Given a template for the Multiset class, implement 4 methods:

* add(self, val): adds val to the multiset
* remove(self, val): if val is in the multiset, removes val from the multiset; otherwise, do nothing
* \_\_contains\_\_(self, val): returns True if val is in the multiset; otherwise, it returns False
* \_\_len\_\_(self): returns the number of elements in the multiset

Additional methods are allowed as necessary.

The implementations of the 4 required methods will be tested by a provided code stub on several input files. Each input file contains *several* operations, each of one of the below types. Values returned by query and size operations are appended to a result list, which is printed as the output by the provided code stub.

* add val: calls add(val) on the Multiset instance
* remove val: calls remove(val) on the Multiset instance
* query val: appends the result of expression val in m, where m is an instance of Multiset, and appends the value of that expression to the result list
* size: calls len(m), where m is an instance of Multiset, and appends the returned value to the result list

Complete the class Multiset in the editor below with the 4 methods given above (add, remove, \_\_contains\_\_, and \_\_len\_\_).

**Constraints**

* 1 ≤ number of operations in one test file ≤ 105
* if val is a parameter of operation, then val is an integer and 1 ≤ val ≤ 109

Input Format Format for Custom Testing

In the first line, there is a single integer, q, denoting the number of queries.

Then, q lines follow. In the ith of them, there is a string denoting an operation and optionally an integer denoting the parameter of the operation.

Sample Case 0

**Sample Input**

STDIN      Function

-----      --------

12 → number of queries, q = 12

query 1 → operations = ["query 1", "add 1", ..., "query 2", "size"]

add 1

query 1

remove 1

query 1

add 2

add 2

size

query 2

remove 2

query 2

size

**Sample Output**

False

True

False

2

True

True

1

**Explanation**

There are 12 operations to be performed. Start with an empty multiset: multiset = [].

1. The first operation asks if 1 is in the multiset. It is not, so False is appended to the result: result = [False].
2. The second operation adds 1 to the multiset: multiset = [1].
3. The third operation asks if 1 is in the multiset. It is now, so True is appended to the result: result = [False, True].
4. The fourth operation removes 1 from the multiset: multiset = [].
5. The fifth operation asks if 1 is in the multiset. It is not, so False is appended to the result: result = [False, True, False].
6. The sixth operation adds 2 to the multiset: multiset = [2].
7. The seventh operation adds 2 to the multiset: multiset = [2, 2].
8. The next operation asks what is the size of the multiset: result = [False, True, False, 2].
9. The next operation asks if 2 is in the multiset. It is, so True is appended to the result: result = [False, True, False, 2, True].
10. The next operation removes 2 from the multiset: multiset = [2]
11. The next operation asks if 2 is in the multiset. It is, so True is appended to the result: result = [False, True, False, 2, True, True].
12. Finally, the last operation asks for the size of the multiset and the length, 1, is appended to the result. result = [False, True, False, 2, True, True, 1]

Sample Case 1

**Sample Input**

STDIN      Function

-----      --------

3 → number of queries, q = 3

size → operations = ["size", "add 17", "size"]

add 17

size

**Sample Output**

0

1

**Explanation**

There are 3 operations to be performed. Start with the empty multiset: multiset = [].

1. The first asks what is the size of the multiset. Since the multiset is empty, 0 is appended to the result: result = [0].
2. The second operation adds 17 to the multiset: multiset = [17].
3. The third operation asks what is the size of the multiset. 1 is appended to the result: result = [0, 1].

Python 3



Autocomplete Ready

1

10

11

12

13

14

15

16

17

18

19

20

#!/bin/python3

class Multiset:

    List = []

    def add(self, val):

        # adds one occurrence of val from the multiset, if any

        self.List.append(val)

    def remove(self, val):

        # removes one occurrence of val from the multiset, if any

        try: self.List.remove(val)



Line: 17 Col: 1

* **Test Results**
* **Custom Input**

Run

Submit Code

### Compiled successfully.

**Run all test cases**

Input (stdin)

* **12**
* **query 1**
* **add 1**
* **query 1**
* **remove 1**
* **query 1**
* **add 2**
* **add 2**
* **size**
* **query 2**
* **remove 2**
* **query 2**
* **size**

Your Output (stdout)

* **False**
* **True**
* **False**
* **2**
* **True**
* **True**
* **1**

Expected Output

* **False**
* **True**
* **False**
* **2**
* **True**
* **True**
* **1**

pass

**Copy/paste is not available in this region. You can copy sample input/output and you can paste code in the editor**

#!/bin/python3

import math

import os

import random

import re

import sys

class Multiset:

    List = []

    def add(self, val):

        # adds one occurrence of val from the multiset, if any

        self.List.append(val)

    def remove(self, val):

        # removes one occurrence of val from the multiset, if any

        try: self.List.remove(val)

        except ValueError: pass

    def \_\_contains\_\_(self, val):

        # returns True when val is in the multiset, else returns False

        return val in self.List

    def \_\_len\_\_(self):

        # returns the number of elements in the multiset

        return len(self.List)

if \_\_name\_\_ == '\_\_main\_\_':

    def performOperations(operations):

        m = Multiset()

        result = []

        for op\_str in operations:

            elems = op\_str.split()

            if elems[0] == 'size':

                result.append(len(m))

            else:

                op, val = elems[0], int(elems[1])

                if op == 'query':

                    result.append(val in m)

                elif op == 'add':

                    m.add(val)

                elif op == 'remove':

                    m.remove(val)

        return result

    q = int(input())

    operations = []

    for \_ in range(q):

        operations.append(input())

    result = performOperations(operations)

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    fptr.write('\n'.join(map(str, result)))

    fptr.write('\n')

    fptr.close()

Input (stdin)

* **12**
* **query 1**
* **add 1**
* **query 1**
* **remove 1**
* **query 1**
* **add 2**
* **add 2**
* **size**
* **query 2**
* **remove 2**
* **query 2**
* **size**

Your Output (stdout)

* **False**
* **True**
* **False**
* **2**
* **True**
* **True**
* **1**

Expected Output

* **False**
* **True**
* **False**
* **2**
* **True**
* **True**
* **1**

# 2. Python: Shape Classes with Area Method

Implement two classes:

Rectangle**:**

* The constructor for Rectangle must take two arguments that denote the lengths of the rectangle's sides.
* The class must have an area method that returns the area of the rectangle.

Circle:

* The constructor for Circle must take one argument that denotes the radius of the circle.
* The Circle class must have an area method that returns the area of the circle. To implement the area method, use a precise Pi value, preferably the constant math.pi.

Your implementation of all the classes will be tested by a provided code stub on several input files. Each input file contains *several* queries, and each query constructs an object of one of the classes and prints the area of this object to the standard output with exactly 2 decimal points.

**Constraints**

* 1 ≤ the number of queries in one test file ≤ 105
* 1 ≤  the value of all parameters passed to construct the objects ≤ 103

Input Format Format for Custom Testing

In the first line, there is a single integer, q, the number of queries.

Then, q lines follow. In the ith of them, there are space-separated parameters. The first of them denotes the shape to be constructed, and the remaining parameters denote the parameters for the constructor.

Sample Case 0

**Sample Input**

STDIN       Function

-----       --------

2 → number of queries, q = 2

circle 1 → query parameters = ["circle 1", "rectangle 2 3"]

rectangle 2 3

**Sample Output**

3.14

6.00

**Explanation**

There are 2 queries. In the first of them, an object of class Circle with radius 1 is constructed. Then, the value of its area property, with exactly 2 decimal points, is printed to the output. Since the radius of the circle is 1, then the printed area is 3.14 (pi \* radius2). In the second query, the object of class Rectangle is constructed with side lengths of 2 and 3. Then, the value of its area property, with exactly 2 decimal points, is printed to the output. Since the side lengths are 2 and 3, then the printed area is 6.00.

Sample Case 1

**Sample Input**

STDIN       Function

-----       --------

3 → number of queries, q = 3

rectangle 5 7 → query parameters = ["rectange 5 7", "rectangle 7 5", "circle 1000"]

rectangle 7 5

circle 1000

**Sample Output**

35.00

35.00

3141592.65

**Explanation**

There are 3 queries. In the first of them, an object of class Rectangle with side lengths of 5 and 7 is constructed. Then, the value of its area property (5 \* 7 = 35), with exactly 2 decimal points, is printed to the output (35.00). The second query likewise returns the same result, since (7 \* 5 = 35). In the third query, an object of class Circle with radius 1000 is constructed. Then, the value of its area property, with exactly 2 decimal points is printed to the output. Since the radius of the circle is 1000, then the printed area is (pi \* 10002) = 3141592.65.

Python 3



Autocomplete Ready

22

23

14

15

16

17

18

19

20

21

1

10

11

12

13

    queries = []

    for \_ in range(q):

class Circle:

    def \_\_init\_\_(self, d): self.areavalue = 3.14\*d

    def area(self): return self.areavalue

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    q = int(input())

#!/bin/python3

class Rectangle:

    def \_\_init\_\_(self, l, w): self.areavalue = l\*w

    def area(self): return self.areavalue



Line: 16 Col: 42

* **Test Results**
* **Custom Input**

Run

Submit Code

### Compiled successfully.2/15 test cases passed

* **Test case 1**
* **Test case 3**
* **Test case 4**
* **Test case 5**
* **Test case 6**
* **Test case 7**
* **Test case 8**
* **Test case 9**
* **Test case 10**
* **Test case 11**
* **Test case 12**
* **Test case 13**
* **Test case 14**
* **Test case 0**
* **Test case 2**

Input (stdin)

Run as Custom Input

|

Download

* **3**
* **rectangle 5 7**
* **rectangle 7 5**
* **circle 1000**

Your Output (stdout)

* **35.00**
* **35.00**
* **3140.00**

Expected Output

Download

* **35.00**
* **35.00**
* **3141592.65**

areavalue

**Copy/paste is not available in this region. You can copy sample input/output and you can paste code in the editor**

#!/bin/python3

import math

import os

import random

import re

import sys

class Rectangle:

    def \_\_init\_\_(self, l, w): self.areavalue = l\*w

    def area(self): return self.areavalue

class Circle:

    def \_\_init\_\_(self, r): self.areavalue = math.pi\*r\*\*2

    def area(self): return self.areavalue

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    q = int(input())

    queries = []

    for \_ in range(q):

        args = input().split()

        shape\_name, params = args[0], tuple(map(int, args[1:]))

        if shape\_name == "rectangle":

            a, b = params[0], params[1]

            shape = Rectangle(a, b)

        elif shape\_name == "circle":

            r = params[0]

            shape = Circle(r)

        else:

            raise ValueError("invalid shape type")

        fptr.write("%.2f\n" % shape.area())

    fptr.close()