**5 kyu**

**Numbers and its Reversal Having Same Prime Factors.**

7085% of 173 of40[raulbc777](https://www.codewars.com/users/raulbc777)

C#

* [TRAIN AGAIN](https://www.codewars.com/kata/numbers-and-its-reversal-having-same-prime-factors/train/csharp)
* [NEXT KATA](https://www.codewars.com/trainer/csharp)

Details

[Solutions](https://www.codewars.com/kata/numbers-and-its-reversal-having-same-prime-factors/solutions/csharp)

[Discourse (17)](https://www.codewars.com/kata/numbers-and-its-reversal-having-same-prime-factors/discuss/csharp)

* Add to Collection
* |
* Share this kata:

The number 1089 is the smallest one, non palindromic, that has the same prime factors that its reversal. Thus,

prime factorization of 1089 with 3, 3, 11, 11 -------> 3, 11

prime factorization of 9801 with 3, 3, 3, 3, 11, 11 -------> 3, 11

The task for this kata is to create a function same\_factRev(), that receives a nMax, to find all the numbers with the above property, bellow nMax.

the function same\_factRev(), will output a sorted list with the found numbers bellow nMax

Let'se some cases

same\_factRev(1100) -----> [1089]

same\_factRev(2500) -----> [1089, 2178]

(Palindromic numbers are like: 171, 454, 4224, these ones should be discarded)

Happy coding!!

(The sequence of these kind of numbers is registered in OEIS as A110819)

<https://www.codewars.com/kata/numbers-and-its-reversal-having-same-prime-factors/csharp>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

// A function to print all prime

// factors of a given number n

public static string primeFactors(int n)

{

HashSet<int> hs = new HashSet<int>();

// Print the number of 2s that divide n

while (n % 2 == 0)

{

//Console.Write(2 + " ");

hs.Add(2);

n /= 2;

}

// n must be odd at this point. So we can

// skip one element (Note i = i +2)

for (int i = 3; i <= Math.Sqrt(n); i += 2)

{

// While i divides n, print i and divide n

while (n % i == 0)

{

//Console.Write(i + " ");

hs.Add(i);

n /= i;

}

}

// This condition is to handle the case whien

// n is a prime number greater than 2

if (n > 2)

{

//Console.Write(n);

hs.Add(n);

}

string ans = "";

foreach(int elem in hs)

{

ans += elem + " ";

}

return ans.Trim();

}

public static int[] SameFactRev(int nMax)

{

// your code

List<int> ans = new List<int>();

for (int i = 1; i <= nMax; i++)

{

//string rev = "";

char[] ich = i.ToString().ToCharArray();

Array.Reverse(ich);

int rev = int.Parse(new string(ich));

//Console.WriteLine(i + " " + rev);

// Console.WriteLine(primeFactors(i) + " - " + primeFactors(rev));

if ( (i.ToString() != rev.ToString()) && primeFactors(i) == primeFactors(rev))

{

//Console.WriteLine(i + " " + rev + " => " + primeFactors(i) + " - " + primeFactors(rev));

ans.Add(i);

}

}

return ans.ToArray();

}

//public static void test1()

//{

// Console.WriteLine("Basic Tests SameFactRev");

// int[] l = { 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000 };

// int[][] r = new int[][] { new int[] {}, new int[] {1089}, new int[] {1089, 2178}, new int[] {1089, 2178}, new int[] {1089, 2178, 4356},

// new int[] {1089, 2178, 4356}, new int[] {1089, 2178, 4356, 6534}, new int[] {1089, 2178, 4356, 6534},

// new int[] {1089, 2178, 4356, 6534, 8712}, new int[] {1089, 2178, 4356, 6534, 8712, 9801} };

// tests(l, r);

//}

// Driver Code

public static void Main()

{

//int n = 315;

//Console.WriteLine( primeFactors(n));

int[] ans = SameFactRev(10000);

//for(int i =0; i<ans.Length; i++)

//{

// Console.Write(ans[i] + " ");

//}

Console.ReadLine();

}

}

}

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package javaapplication9;

import java.util.ArrayList;

import java.util.HashSet;

/\*\*

\*

\* @author Usuario

\*/

public class JavaApplication9 {

// A function to print all prime

// factors of a given number n

public static String primeFactors(int n)

{

//HashSet<int> hs = new HashSet<int>();

HashSet<Integer> hs = new HashSet();

// Print the number of 2s that divide n

while (n % 2 == 0)

{

//Console.Write(2 + " ");

hs.add(2);

n /= 2;

}

// n must be odd at this point. So we can

// skip one element (Note i = i +2)

for (int i = 3; i \* i <= n; i += 2)

{

// While i divides n, print i and divide n

while (n % i == 0)

{

//Console.Write(i + " ");

hs.add(i);

n /= i;

}

}

// This condition is to handle the case whien

// n is a prime number greater than 2

if (n > 2)

{

//Console.Write(n);

hs.add(n);

}

String ans = "";

/\*

foreach(int elem in hs)

{

ans += elem + " ";

}\*/

for(int elem : hs) {

ans += elem + " ";

}

return ans.trim();

}

public static int[] sameFactRev(int nMax)

{

// your code

ArrayList<Integer> ans = new ArrayList();

for (int i = 1; i <= nMax; i++)

{

//string rev = "";

// char[] ich = i.ToString().ToCharArray();

String istr = String.valueOf(i);

//Array.Reverse(ich);

String reverse = "";

for(int j = istr.length() - 1; j >= 0; j--)

{

reverse = reverse + istr.charAt(j);

}

int rev = Integer.parseInt(reverse);

//int rev = int.Parse(new string(ich));

//Console.WriteLine(i + " " + rev);

// Console.WriteLine(primeFactors(i) + " - " + primeFactors(rev));

if ( (!istr.equals(reverse)) && primeFactors(i).equals(primeFactors(rev)))

{

//Console.WriteLine(i + " " + rev + " => " + primeFactors(i) + " - " + primeFactors(rev));

ans.add(i);

}

}

//return ans.ToArray();

int[] arr = new int[ans.size()];

for(int i =0; i< arr.length; i++) {

arr[i] = ans.get(i);

}

return arr;

}

public static void main(String[] args) {

// TODO code application logic here

}

}