|  |
| --- |
| public class Task1 { |
|  | public static void main(String[] args) { |
|  | int[] Numbers = new int[args.length]; |
|  | for (int i = 0; i < args.length; i++) { |
|  | Numbers[i] = Integer.parseInt(args[i]); |
|  | } |
|  | System.out.println(solutions(Numbers[0], Numbers[1], Numbers[2])); |
|  | } |
|  |  |
|  |  |
|  | public static int solutions(int a, int b, int c) { |
|  | double result = Math.pow(b, 2) - 4 \* a \* c; |
|  | if (result > 0) { |
|  | return 2; |
|  | } |
|  | else if (result == 0) { |
|  | return 1; |
|  | } |
|  | return 0; |
|  | } |
|  | } |

|  |
| --- |
| public class Task2 { |
|  | public static void main(String[] args) { |
|  | String s = ""; |
|  | for (int i = 0; i < args.length; i++) { |
|  | s += args[i] + " "; |
|  | } |
|  | System.out.println(findZip(s)); |
|  | } |
|  |  |
|  |  |
|  | public static int findZip(String s) { |
|  | if (s.indexOf("zip") != -1 && s.lastIndexOf("zip") != s.indexOf("zip")) { |
|  | return s.lastIndexOf("zip"); |
|  | } |
|  | return -1; |
|  | } |
|  | } |

|  |
| --- |
| public class Task3 { |
|  | public static void main(String[] args) { |
|  | System.out.println(checkPerfect(Integer.parseInt(args[0]))); |
|  | } |
|  |  |
|  |  |
|  | public static boolean checkPerfect(int n) { |
|  | int result = 0; |
|  | for (int i = 1; i < n; i++) { |
|  | if (n % i == 0) { |
|  | result += i; |
|  | } |
|  | if (result > n) { |
|  | return false; |
|  | } |
|  | } |
|  | return (result == n); |
|  | } |
|  | } |

|  |
| --- |
| public class Task4 { |
|  | public static void main(String[] args) { |
|  | System.out.println(checkPerfect(args[0])); |
|  | } |
|  |  |
|  |  |
|  | public static String checkPerfect(String s) { |
|  | if (s.length() <= 2) { |
|  | return "Incompatible."; |
|  | } |
|  | if (s.charAt(0) == s.charAt(s.length() - 1)) { |
|  | return "Two's a pair."; |
|  | } |
|  | char first = s.charAt(0); |
|  | char last = s.charAt(s.length() - 1); |
|  | char[] chars = s.toCharArray(); |
|  | chars[0] = last; |
|  | chars[s.length() - 1] = first; |
|  | return String.valueOf(chars); |
|  | } |
|  | } |

|  |
| --- |
| import java.util.Arrays; |
|  | import java.util.List; |
|  |  |
|  |  |
|  |  |
|  | public class Task5 { |
|  | public static void main(String[] args) { |
|  | System.out.println(isValidHex(args[0])); |
|  | } |
|  |  |
|  |  |
|  | public static boolean isValidHex(String s) { |
|  | String[] allowed = new String[] {"1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "A", "B", "C", "D", "E", "F"}; |
|  | List<String> list = Arrays.asList(allowed); |
|  | if (s.length() != 7 || Character.toString(s.charAt(0)) != "#") { |
|  | return false; |
|  | } |
|  | for (int i = 1; i < s.length(); i++) { |
|  | if (list.contains(Character.toString(s.charAt(i)).toUpperCase())) { |
|  | continue; |
|  | } |
|  | else { |
|  | return false; |
|  | } |
|  | } |
|  | return true; |
|  | } |
|  | } |

|  |
| --- |
| import java.util.Arrays; |
|  | import java.util.List; |
|  | import java.util.ArrayList; |
|  | import java.util.HashSet; |
|  | import java.util.\*; |
|  |  |
|  |  |
|  | public class Task6 { |
|  | public static void main(String[] args) { |
|  | boolean old = true; |
|  | Set<Integer> arr1 = new HashSet<Integer>(); |
|  | Set<Integer> arr2 = new HashSet<Integer>(); |
|  | for (int i = 0; i < args.length; i++) { |
|  | if (args[i].equals(".")) { |
|  | old = false; |
|  | continue; |
|  | } |
|  | if (old) arr1.add(Integer.parseInt(args[i])); |
|  | else arr2.add(Integer.parseInt(args[i])); |
|  | } |
|  | System.out.println(arr1.size() == arr2.size()); |
|  | } |
|  | } |

|  |
| --- |
| import java.util.Arrays; |
|  | import java.util.List; |
|  |  |
|  |  |
|  |  |
|  | public class Task7 { |
|  | public static void main(String[] args) { |
|  | System.out.println(isKaprekar(Integer.parseInt(args[0]))); |
|  | } |
|  |  |
|  |  |
|  | public static boolean isKaprekar(int x) { |
|  | String result = String.valueOf((int)Math.pow(x, 2)); |
|  | int num1; |
|  | int num2; |
|  | if (result.length() == 1) num1 = 0; |
|  | else num1 = Integer.parseInt(result.substring(0, (int)(result.length() / 2))); |
|  | num2 = Integer.parseInt(result.substring((int)(result.length() / 2))); |
|  | System.out.printf("%s, %s \n", num1, num2); |
|  | return (num1 + num2 == x); |
|  | } |
|  | } |

|  |
| --- |
| public class Task8 { |
|  | public static void main(String[] args) { |
|  | System.out.println(lognestZero(args[0])); |
|  | } |
|  |  |
|  |  |
|  | public static String lognestZero(String s) { |
|  | int count = 0; |
|  | int temp = 0; |
|  | String result = ""; |
|  | for (int i = 0; i < s.length(); i++) { |
|  | System.out.printf("count: %s, temp: %s, i: %s \n", count, temp, i); |
|  | //System.out.println((int)(s.charAt(i)) == 0); |
|  | //System.out.println(Character.toString(s.charAt(i)) == "0"); |
|  | //System.out.println(Character.toString(s.charAt(i)).equals("0")); |
|  | if (Character.toString(s.charAt(i)).equals("0")) temp++; |
|  | else temp = 0; |
|  | if (temp > count) count=temp; |
|  | } |
|  | for (int i = 0; i < count; i++) result += "0"; |
|  | return result; |
|  | } |
|  | } |
| public class Task9 { |
|  | public static void main(String[] args) { |
|  | System.out.println(nextPrime(Integer.parseInt(args[0]))); |
|  | } |
|  |  |
|  |  |
|  | public static int nextPrime(int number) { |
|  | boolean isPrime = true; |
|  | int next = number; |
|  | while (true) { |
|  | isPrime = true; |
|  | for (int i = 2; i < next - 1; i++) { |
|  | if (next % i == 0) isPrime = false; |
|  | } |
|  | if (isPrime) return next; |
|  | next++; |
|  | } |
|  | } |
|  | } |

|  |
| --- |
| import java.util.\*; |
|  |  |
|  |  |
|  | public class Task10 { |
|  | public static void main(String[] args) { |
|  | System.out.println(rightTri(Integer.parseInt(args[0]), Integer.parseInt(args[1]), Integer.parseInt(args[2]))); |
|  | } |
|  |  |
|  |  |
|  | public static boolean rightTri(int x, int y, int z) { |
|  | int[] arr = {x, y, z}; |
|  | Arrays.sort(arr); |
|  | return (Math.sqrt(Math.pow(arr[0], 2) + Math.pow(arr[1], 2)) == arr[2]); |
|  | } |
|  | } |