/\*

\*

\*\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

\* Other Syntax and notes

\* \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

\*/

/\*

\* Decrementing for loop.

\* This allows the collection being looped through to be modified (values added/removed)

\* without crashing the loop!

\*/

for(int i = arrayList.size()-1; i >= 0; i--)

{

arrayList.remove(i);

}

/\*

\* for-each loop syntax

\* add up all the values in this sets answerList and add the final

\* result to the answer array.

\* The list cannot be generic! It must be declared to hold a value type.

\* for-each loop is essentially the same as

\* for(int i = 0; i < arrayList.size(); i++)

\* {

\* String value1 = stringList.get(i);

\* System.out.println(value1);

\* }

\* It simply automatically casts the value to a new instance of String.

\*/

ArrayList<String> stringList = new ArrayList<String>();

for(String value1 : stringList)

{

System.out.println(value1);

}

/\*

\* StringBuffer syntax

\* StringBuffers concatenate and split MUCH faster than a String.

\* If you have to continually add on to a string, use a StringBuffer instead.

\* Just about any object type can be concatenated, ints, floats, strings, etc.

\* It DOES NOT have a .contains method!

\*/

String string1 = "asdf";

StringBuffer buffer = new StringBuffer(); //initialize

buffer.append(string1); // concatenate value

String string2 = buffer.toString(); // convert to string

buffer.reverse(); // reverse order of characters

buffer.deleteCharAt(1); // delete at a int point

string2 = buffer.substring(2,3); // split at a int point

/\*

\* odd-roots (cubed root, fifth root, etc.)

\* The root of any number x can be rewritten as x^1/k, where k = the root

\* It's very important NOT to approximate the decimal value (.333333~)

\* and to cast all values as double.

\*/

int value2 = 60;

double double2 = Math.pow((double)value2, (double)(1.0/3.0));

/\*

\* Round to 2 decimal places (100 = .00, 1000 = .000, etc.)

\* (both decimal values must be the same, as in NOT (value\*100)/10)

\*/

double value = 6.8705933;

double doub = (double)Math.round(value\*100)/100;

System.out.println(doub);

/\*

\* A more ‘proper’ way of doing this is the

\* DecimalFormat class in java.text.DecimalFormat

\* twoDecimal.format() formats the decimal to the appropriate decimal point

\* Double.valueOf() converts the Double object returned to a double value

\* String.format(format, value) ensures that two decimal places will always be printed

\* equally, "%.3f" ensures three decimal places, etc.

\*/

DecimalFormat leadingZeros = **new** DecimalFormat("00.00");

DecimalFormat twoDecimal = **new** DecimalFormat("#.##");

Double doub = 1.005;

Double formatted = Double.valueOf(leadingZeros.format(doub)) // == 01.0

Double formatted2 = Double.valueOf(twoDecimal.format(doub)) // == 1.01

System.out.println(String.format(“%.5f”,formatted)) // 01.00000

System.out.println(String.format(“%.3f”,formatted)) // 1.010

/\*

\* Copy two-dimensional arrays. You cannot use ‘=’ or Arrays.clone!

\* To check for equality, use Arrays.deepEquals(1,2)

\*/

private static int[][] copy2d(int[][] nums)

{

int[][] copy = new int[nums.length][];

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

int[] member = new int[nums[i].length];

System.*arraycopy*(nums[i], 0, member, 0, nums[i].length);

copy[i] = member;

}

return copy;

}

/\*

\* Find all permutations of a given string

\*/

Static ArrayList<String> tempList = new ArrayList<String>();

public static void permuteString(String beginningString, String endingString) {

if (endingString.length() <= 1)

{

*tempList*.add(beginningString + endingString);

}

else{

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

try {

String newString = endingString.substring(0, i) + endingString.substring(i + 1);

*permuteString*(beginningString + endingString.charAt(i), newString);

} catch (StringIndexOutOfBoundsException exception) {

exception.printStackTrace();

}

}

}

/// returned can then be converted to a set to remove any duplicates

Set<String> hs = new LinkedHashSet<String>(*tempList*);

*tempList*.clear();

*tempList*.addAll(hs);

/\*

\* Finding the max/min value in an array

\* There are two ways of going about this, depending on the problem requirements

\*/

/// #1: The simplest, fastest method

ArrayList<Integer> tempList = new ArrayList<Integer>();

Collections.sort(tempList); // minimum value will be at index 0 and max at tempList.size() -1

///#2: Iteration. This will be useful in situations where you cant sort the list for some reason

///Integer.Min\_VALUE guarantees that any value found will be higher, and thus

/// this value will be reset at the first run of the loop

/// if you were looking for the lowest value in the list, use Integer.MAX\_VALUE and flip the > sign

int currentLowest = Integer.*MIN\_VALUE*;

int currentLowestIndex = -1;

for(int i = 0; i < *pointsList*.size(); i++)

{

if(*pointsList*.get(i) > currentLowest)

{

currentLowest = *pointsList*.get(i);

currentLowestIndex = i;

}

}

/\*

\* Look-up tables and finding max/min values

\* Look-up tables are an easier alterative to multi-dimensional arrays, particularly

\* when you get into 3+ dimensions. However, they are not as efficient, so be warned.

\* Like a mutli-dimensional array, they allow you to track information related to

\* something by cross-referencing indices with other arrays.

\*/

//For this, we would usually need a 4-dimensional array, and iterations would be painful.

// Instead, we use four arraylists.

static ArrayList<String> *teamList* = new ArrayList<String>();

static ArrayList<Integer> *pointList* = new ArrayList<Integer>();

static ArrayList<Integer> *penPointsList* = new ArrayList<Integer>();

static ArrayList<Integer> *answerList* = new ArrayList<Integer>();

// Ensure that each list is added to in the proper order, so that

// the index of (teamName) == index of (points), etc., for the same team

*teamList*.add(teamName);

*pointList*.add(points);

*penPointsList*.add(penPoints);

*answerList*.add(totalPoints);

// Now for the beauty. In finding min/max values (above), if two teams have an equal score, usually

// there is some other tag that identifies the winning team that requires referencing other related

// information, in this case penalty points. To the above min/max code, add this else if.

// Using the same i, we can reference yet another value of the same team and compare to the current ‘winner’

else if(*pointList*.get(i) == currentLowest)

{

if(*penPointsList*.get(i) < *penPointsList*.get(currentLowestIndex))

{

currentLowest = *pointList*.get(i);

currentLowestIndex = i;

}

}

// and finally output references

System.*out*.println(*teamList*.get(currentLowestIndex) + " " + *pointList*.get(currentLowestIndex) + " " +

*answerList*.get(currentLowestIndex));