```
1  namespace ResumenFunciones
2  {
3      public class Colecciones
4      {
5          // FUNCION que devuelve el numero de los valores de una lista ↵
             mayores de 0.
6          public static int GetGreaterThanZero(List<double> list)
7          {
8              if (list == null || list.Count == 0)
9                  return 0;
10
11             int result = 0;
12
13             for (int i = 0; i < list.Count; i++)
14             {
15                 if (list[i] > 0)
16                     result++;
17             }
18             return result;
19         }
20
21         // FUNCION que devuelve el numero de los valores de un array ↵
             mayores de 0.
22         public static int GetGreaterThanZero(double[] array)
23         {
24             if (array == null || array.Length == 0)
25                 return 0;
26
27             int result = 0;
28
29             for (int i = 0; i < array.Length; i++)
30             {
31                 if (array[i] > 0)
32                     result++;
33             }
34             return result;
35         }
36
37         // FUNCION que devuelve true si una lista contiene un numero ↵
             dado.
38         public static bool ContainsNumber(List<int> list, int number)
39         {
40             if (list == null)
41                 return false;
42
43             for (int i = 0; i < list.Count; i++)
44             {
45                 if (list[i] == number)
46                     return true;
47             }
48             return false;
49         }
50
```

```
51          // Funcion que devuelve el valor máximo de una lista.
52          public static int GetMajor(List<int> list)
53          {
54              if (list == null || list.Count == 0)
55                  return int.MinValue;
56
57              int result = list[0];
58              for (int i = 1; i < list.Count; i++)
59              {
60                  if (list[i] > result)
61                      result = list[i];
62              }
63              return result;
64          }
65
66          // Funcion que devuelve el valor mínimo de una lista.
67          public static int GetMinor(List<int> list)
68          {
69              if (list == null || list.Count == 0)
70                  return 0;
71
72              int result = list[0];
73              for (int i = 1; i < list.Count; i++)
74              {
75                  if (list[i] < result)
76                      result = list[i];
77              }
78              return result;
79          }
80
81          // FUNCION que devuelve el índice del numero mayor de una       ↵
              lista.
82          public static int GetIndexMajor(List<int> list)
83          {
84              if (list == null || list.Count == 0)
85                  return -1;
86
87              int index = 0;
88
89              for (int i = 0; i < list.Count; i++)
90              {
91                  if (list[i] == GetMajor(list))
92                      index = i;
93              }
94              return index;
95          }
96
97          // FUNCION que devuelve el índice del numero mayor de una lista ↵
               V2.0.
98          public static int GetIndexMajorV2(List<int> list)
99          {
100             if (list == null || list.Count == 0)
101                 return -1;
```

```
102
103             int index = 0;
104             int aux = list[0];
105             for (int i = 0; i < list.Count; i++)
106             {
107                 if (list[i] > aux)
108                 {
109                     index = i;
110                     aux = list[i];
111                 }
112             }
113             return index;
114         }
115
116         // FUNCION que devuelve true si una lista está ordenada de forma ascendente.
117         // TODO: [EXAMEN] Función que devuelve un booleano si está ordenada.
118         public static bool IsOrdered(List<int> list)
119         {
120             if (list == null || list.Count == 0)
121                 return false;
122
123             bool result = true;
124             for (int i = 0; i < list.Count - 1; i++)
125             {
126                 if (list[i] > list[i + 1])
127                     result = false;
128             }
129             return result;
130         }
131
132         // FUNCION que realiza un SWAP entre elementos de la lista.
133         // TODO: [EXAMEN] Funcion que realiza un SWAP.
134         public static void Swap(List<int> list)
135         {
136             int aux;
137
138             aux = list[0];
139             list[0] = list[1];
140             list[1] = aux;
141         }
142
143         // FUNCION que ordena una lista de forma ascendente.
144         // TODO: [EXAMEN] Ordenar una lista de forma ascendente.
145         public static List<int>? Sort(List<int> list)
146         {
147             if (list == null || list.Count == 0)
148                 return null;
149
150             int aux;
151             for (int i = 0; i < list.Count - 1; i++)
152             {
```

```
153                 for (int j = i + 1; j < list.Count; j++)
154                 {
155                     if (list[i] > list[j])
156                     {
157                         aux = list[i];
158                         list[i] = list[j];
159                         list[j] = aux;
160                     }
161                 }
162             }
163             return list;
164         }
165 
166         // FUNCION que ordena una lista de forma ascendente.
167         // TODO: [EXAMEN] Ordenar una lista de forma ascendente V2.0.
168         public static List<int>? SortV2(List<int> list)
169         {
170             if (list == null || list.Count == 0)
171                 return null;
172 
173             int aux;
174             for (int i = 0; i < list.Count - 1; i++)
175             {
176                 for (int j = list.Count - 1; j > 0; j--)
177                 {
178                     if (list[j] < list[j - 1])
179                     {
180                         aux = list[j - 1];
181                         list[j - 1] = list[j];
182                         list[j] = aux;
183                     }
184                 }
185             }
186             return list;
187         }
188 
189         // FUNCION Binary Search con listas.
190         // TODO: [EXAMEN] Binary Search.
191         public static bool BinarySearch(List<int> list, int number)
192         {
193             if (list == null || list.Count == 0)
194                 return false;
195 
196             int minPosition = 0;
197             int maxPosition = list.Count - 1;
198             int midPosition;
199 
200             while (minPosition <= maxPosition)
201             {
202                 midPosition = (minPosition + maxPosition) / 2;
203 
204                 if (list[midPosition] == number)
205                     return true;
```

```
206
207                 if (number > list[midPosition])
208                     minPosition = midPosition + 1;
209                 else
210                     maxPosition = midPosition - 1;
211             }
212             return false;
213         }
214
215         // FUNCION Binary search con arrays.
216         // TODO: [EXAMEN] Binary Search.
217         public static bool BinarySearch(double[] array, double number)
218         {
219             if (array == null || array.Length == 0)
220                 return false;
221
222             int minPosition = 0;
223             int maxPosition = array.Length - 1;
224             int midPosition;
225
226             while (minPosition <= maxPosition)
227             {
228                 midPosition = (minPosition + maxPosition) / 2;
229
230                 if (array[midPosition] == number)
231                     return true;
232
233                 if (number > array[midPosition])
234                     minPosition = midPosition + 1;
235                 else
236                     maxPosition = midPosition - 1;
237             }
238             return false;
239         }
240
241         // FUNCION que devuelve el valor medio de un array.
242         public static double GetAverage(double[] array)
243         {
244             if (array == null)
245                 return double.NaN;
246
247             if (array.Length == 0)
248                 return 0.0;
249
250             double aux = 0.0;
251
252             for (int i = 0; i < array.Length; i++)
253             {
254                 aux += array[i];
255             }
256             return aux / array.Length;
257         }
258
```

```
259         // FUNCION que devuelve el mayor medio de los elementos de un array que superen el threshold.
260         public static double GetThresholdAverage(double[] array, double Threshold)
261         {
262             if (array == null)
263                 return double.NaN;
264 
265             if (array.Length == 0)
266                 return 0.0;
267 
268             double aux = 0.0;
269             int count = 0;
270 
271             for (int i = 0; i < array.Length; i++)
272             {
273                 if (array[i] > Threshold)
274                 {
275                     aux += array[i];
276                     count++;
277                 }
278             }
279 
280             if (count == 0)
281                 return 0.0;
282 
283             return aux / count;
284         }
285 
286         // FUNCION que devuelve el numero de repeticiones de valores de una lista.
287         public static int GetMaxNumberRepeated(List<int> list)
288         {
289             if (list == null || list.Count == 0)
290                 return 0;
291 
292             int result = 0;
293             int maxNumber = GetMajor(list);
294 
295             for (int i = 0; i < list.Count; i++)
296             {
297                 if (list[i] == maxNumber)
298                     result++;
299             }
300             return result;
301         }
302 
303         // FUNCION que devuelve una lista ordenada de forma inversa.
304         public static List<int>? GetReverseList(List<int> list)
305         {
306             if (list == null || list.Count == 0)
307                 return null;
308 
```

```
309             List<int> result = new List<int>();
310 
311             for (int i = list.Count - 1; i >= 0; i--)
312             {
313                 result.Add(list[i]);
314             }
315             return result;
316         }
317 
318         // FUNCION que devuelve un array ordenado de forma inversa.
319         public static int[] GetReverseArray(int[] array)
320         {
321             int[] result = new int[array.Length];
322             int auxiliar = 0;
323 
324             for (int i = array.Length - 1; i >= 0; i--)
325             {
326                 result[auxiliar] = array[i];
327                 auxiliar++;
328             }
329             return result;
330         }
331 
332         // FUNCION que añade un valor a un array.
333         // TODO: [EXAMEN] Añadir un valor a un array.
334         public static int[] AddValuetoArray(int[] array, int number)
335         {
336             int[] result;
337             if (array == null)
338             {
339                 result = new int[1] { number };
340                 return result;
341             }
342 
343             result = new int[array.Length + 1];
344 
345             for (int i = 0; i < array.Length; i++)
346             {
347                 result[i] = array[i];
348             }
349             result[array.Length] = number;
350 
351             return result;
352         }
353 
354         // FUNCION que elimina un valor que le paso por parametros de la lista.
355         // TODO: [EXAMEN] Funcion que elimina el valor que ocupa un valor dado 'index'.
356         public static void RemoveAt(List<int> list, int index)
357         {
358             if (list == null ||index < 0)
359                 return;
```

```
360
361             for (int i = 0; i < list.Count; i++)
362             {
363                 if (list[i] == index)
364                 {
365                     list.RemoveAt(i);
366                     i--;
367                 }
368             }
369         }
370
371         // FUNCION 'RemoveAt' que utiliza una instruccion 'BREAK'.
372         public static void RemoveElementBreak(List<int> list, int
      value)
373         {
374             if (list == null)
375                 return;
376
377             for (int i = 0; i < list.Count; i++)
378             {
379                 if (list[i] == value)
380                 {
381                     list.RemoveAt(i);
382                     i--;
383                     break;
384                 }
385             }
386         }
387
388         // FUNCION que borra un elemento de un array.
389         // TODO: [EXAMEN] Borrar un elemento de un array.
390         public static int[]? RemoveValueFromArray(int[] array, int
      value)
391         {
392             if (array == null || array.Length == 0)
393                 return null;
394
395             int repeatedValues = 0;
396             for(int i = 0; i < array.Length; i++)
397             {
398                 if (array[i] == value)
399                     repeatedValues++;
400             }
401
402             int newLength = array.Length - repeatedValues;
403             int[] result = new int[newLength];
404
405             for(int i = 0; i < newLength; i++)
406             {
407                 if (array[i] != value)
408                     result[i] = array[i];
409             }
410             return result;
```

```
411         }
412
413         // FUNCION que elimina de una lista, una lista de valores.
414         // TODO: [EXAMEN] Eliminar valores de una lista.
415         public static void RemoveValues(List<int> list, List<int>
              listValues)
416         {
417             if (list == null)
418                 return;
419
420             for (int j = 0; j < listValues.Count; j++)
421             {
422                 for (int i = 0; i < list.Count; i++)
423                 {
424                     if (list[i] == list[j])
425                         list.RemoveAt(i--);
426                 }
427             }
428         }
429
430         // FUNCION que elimina de una lista, una lista de valores V2.0.
431         public static void RemoveValuesV2(List<int> list, List<int>
              listValues)
432         {
433             if (list == null)
434                 return;
435
436             for (int i = 0, j = 0; i < listValues.Count; i++)
437             {
438                 while (j < listValues.Count)
439                 {
440                     if (list[i] == listValues[j])
441                         list.RemoveAt(i--);
442                     j++;
443                 }
444                 j = 0;
445             }
446         }
447         // FUNCION que cuenta los elementos de un array.
448         public static int CountArray(int[] array, int value)
449         {
450             int count = 0;
451
452             for (int i = 0; i < array.Length; i++)
453             {
454                 if (IsDequal(array[i], value))
455                     count++;
456             }
457             return count;
458         }
459
460         // FUNCION que crea un array nuevo.
461         public static void MakeArray(int[] array, int value)
```

```
462         {
463             int capacity = CountArray(array, value);
464             int[] result = new int[capacity];
465             for (int i = 0; i < capacity; i++)
466                 if (IsDequal(value, array[i]))
467                     result[i] = value;
468         }
469 
470         // FUNCION que devuelve un booleano si dos numeros son iguales.
471         public static bool IsDequal(int number1, int number2)
472         {
473             return (number1 != number2);
474         }
475 
476         // FUNCION que imprime Listas en consola.
477         public static void PrintList(List<int> list)
478         {
479             for (int i = 0; i < list.Count; i++)
480             {
481                 Console.WriteLine(list[i]);
482             }
483         }
484 
485         // FUNCION que imprime Arrays en consola.
486         public static void PrintList(int[] array)
487         {
488             for (int i = 0; i < array.Length; i++)
489             {
490                 Console.WriteLine(array[i]);
491             }
492         }
493     }
494 }
```