C# MASH-UP

C# TIPS & TRICKS, QUIZZ, ETC.

PROBLEM: BANK STYLE ROUNDING

```
1 Console.WriteLine(Math.Round(1.5, MidpointRounding.ToEven));
2 Console.WriteLine(Math.Round(2.5, MidpointRounding.ToEven));
```

LinqPad

PROBLEM: POINT VS PEN

```
1  var point1 = new Point(5,5);
2  var point2 = point1;
3  point2.X = 50;
4
5  var pen1 = new Pen(Color.Green);
6  var pen2 = pen1;
7  pen2.Color = Color.Red;
8
9  Console.WriteLine(point1.X);
10 Console.WriteLine(point2.X);
11
12 Console.WriteLine(pen1.Color.ToString());
13 Console.WriteLine(pen2.Color.ToString());
```

LinqPad

PROBLEM: POINT VS PEN

```
1  class Program
2  {
3      static Point point;
4      static Pen pen;
5      static void Main(string[] args)
6      {
7          Console.WriteLine(point == null);
8          Console.WriteLine(pen == null);
9      }
10 }
```

LinqPad

PROBLEM: STRING COMPARISON

```
1 string s = "s      e";
2 string s1 = "s     e";
3
4 // 7, 6
5 Console.WriteLine($"{s.Length}, {s1.Length}");
6 // False
7 Console.WriteLine(s == s1);
8 // True
9 Console.WriteLine(s.Equals(s1, StringComparison.InvariantCultureIgnoreCase));
```

LinqPad

PROBLEM: STRING COMPARISON

```
1 string s = "strasse";
2 string s1 = "straße";
3
4 // 7, 6
5 Console.WriteLine($"{s.Length}, {s1.Length}");
6 // False
7 Console.WriteLine(s == s1);
8 // True
9 Console.WriteLine(s.Equals(s1, StringComparison.InvariantCultureIgnoreCase));
```

Q: VARIABLE CAPTURING

```
1  var actions = new List<Action>();
2  for (int i = 0; i < 3; i++)
3  {
4      actions.Add(() => Console.Write(i));
5  };
6
7  foreach (var action in actions)
8  {
9      action();
10 }
```

LinqPad

A: VARIABLE CAPTURING

```
1  var actions = new List<Action>();
2  for (int i = 0; i < 3; i++)
3  {
4      var j = i;
5      actions.Add(() => Console.Write(j));
6  };
7
8  foreach (var action in actions)
9  {
10     action();
11 }
```

LinqPad

Q: LINQ & LISTS

```
1 var list = new List<string> { "A1", "B1", "B2"};
2 var filterredList = list.Where(i => i.StartsWith("B"));
3 list.Remove("B1");
4 Console.WriteLine(filterredList.Count());
```

LinqPad

A: LINQ & LISTS

```
1 var list = new List<string> { "A1", "B1", "B2"};
2 var filterredList = list.Where(i => i.StartsWith("B")).ToList();
3 list.Remove("B1");
4 Console.WriteLine(filterredList.Count());
```

LinqPad

Q: ENUMERATOR

```
1 var i = new
2 {
3     Items = new List<int> {1,2,3}.GetEnumerator()
4 };
5
6 while (i.Items.MoveNext())
7 {
8     Console.WriteLine(i.Items.Current);
9 }
```

LinqPad

A: ENUMERATOR

```
1  var i = new
2  {
3      Items = new List<int> {1,2,3}.GetEnumerator()
4  };
5
6  var enumerator = i.Items;
7  while (enumerator.MoveNext())
8  {
9      Console.WriteLine(enumerator.Current);
10 }
```

LinqPad

Q: NOTHING VS NOTHING

```
1  var sequenceA0 = new int[2] { 1, 2 };
2  var sequenceB0 = new int[2] { 1, 2 };
3  Console.WriteLine(sequenceA0 == sequenceB0);
4
5  var sequenceA1 = new int[2] { 2, 1 };
6  var sequenceB1 = new int[2] { 1, 2 };
7  Console.WriteLine(sequenceA1 == sequenceB1);
8
9  var sequenceA2 = new int[0];
10 var sequenceB2 = new int[0];
11 Console.WriteLine(sequenceA2 == sequenceB2);
```

LinqPad

A: NOTHING VS NOTHING

```
1  var sequenceA0 = new int[2] { 1, 2 };
2  var sequenceB0 = new int[2] { 1, 2 };
3  Console.WriteLine(sequenceA0.SequenceEqual(sequenceB0));
4
5  var sequenceA1 = new int[2] { 2, 1 };
6  var sequenceB1 = new int[2] { 1, 2 };
7  Console.WriteLine(sequenceA1.SequenceEqual(sequenceB1));
8
9  var sequenceA2 = new int[0];
10 var sequenceB2 = new int[0];
11 Console.WriteLine(sequenceA2.SequenceEqual(sequenceB2));
```

LinqPad

Q: FIND AND LOG

```
1 var list = new List<int> {1, 2, 3, 4, 5};
2 var filterred = list.FindAll(i =>
3     {
4         // Log filterred value
5         Console.Write(i);
6         return i < 3;
7     });
8 Console.WriteLine(string.Join("", filterred));
```

LinqPad

A: FIND AND LOG

```
1  var list = new List<int> {1, 2, 3, 4, 5};
2  var filterred = list.FindAll(i =>
3      {
4          if (i < 3)
5          { // Log filterred value
6              Console.Write(i);
7              return true;
8          }
9          return false;
10     });
11 Console.WriteLine(string.Join("", filterred));
```

LinqPad

Q: TRY FINALLY

```
 1 int Test()
 2 {
 3     try
 4     {
 5         return 1;
 6     }
 7     finally
 8     {
 9         return 2;
10     }
11 };
12
13 Console.WriteLine(Test());
```

LinqPad

A: TRY FINALLY

```
 6 {
 7     public static void main(String[] args)
 8     {
 9
10         System.out.println("Finally win!" + Test());
11     }
12
13     static int Test()
14     {
15         try {return 1;}
16         finally {return 2;}
17     };
18 }
```

```
Finally win!2
```

LinqPad

PROBLEM: YIELD RETURN

```
1  IEnumerable<string> Foo()
2  {
3      yield return "1";
4      yield return "2";
5      Console.WriteLine("3");
6  }
7
8  foreach (var s in Foo())
9  {
10     Console.WriteLine(s);
11 }
```

LinqPad

PROBLEM: FLOATING ENUM

```
1  enum MyEnum { Hello, Hola };
2
3  static void Main()
4  {
5      MyEnum f0 = 0.0f;
6      Console.WriteLine(f0);
7
8      MyEnum f1 = 1.0f;
9      Console.WriteLine(f1);
10 }
```

LinqPad

Q: NULL + NULL

```
1 var s = ((string)null + null);
2 var isNull = (s == null);
3 Console.WriteLine(isNull);
```

LinqPad

Q: NULL + NULL

```
public static string Concat(string str0, string str1)
{
  if (string.IsNullOrEmpty(str0))
  {
    if (string.IsNullOrEmpty(str1))
      return string.Empty;
    return str1;
  }
  if (string.IsNullOrEmpty(str1))
    return str0;
  int length = str0.Length;
  string dest = string.FastAllocateString(length + str1.Length);
  string.FillStringChecked(dest, 0, str0);
  string.FillStringChecked(dest, length, str1);
  return dest;
}
```

PROBLEM: ADDITION SEQUENCE

```
1 Console.WriteLine(1 + 2 + "A");
2 Console.WriteLine("A" + 1 + 2);
3 Console.WriteLine('A' + 1 + 2);
4 Console.WriteLine(1 + 2 + 'A');
```

LinqPad

PROBLEM: YIELD EXCEPTION

```
1  public static IEnumerable<int> GetRawData()
2  {
3      yield return 1;
4      throw new Exception();
5      yield return 2;
6  }
7  void Main()
8  {
9      var raw = GetRawData();
10     var nums = raw.Take(2).Select(i => i * 1);
11     Console.WriteLine(nums.FirstOrDefault());
12 }
```

LinqPad

PROBLEM: BITWISE SHIFT

```
1 int i = 1 << 1;
2 int j = 1 << 33;
3 Console.WriteLine($"i:{i}, j:{j}");
```

LinqPad

PROBLEM: BITWISE SHIFT EXPLANATION

```
1 int i = 1 << 1;
2 int j = 1 << (33 & 0x11111);
3 Console.WriteLine(i == j);
```

LinqPad

PROBLEM: ENUM PARSE

```
1  public enum MyEnum
2  {
3      V1, V2, V3, V4, V5, V6, V7, V8
4  }
5  void Main()
6  {
7      MyEnum result;
8      var str = "V1, V2, V3";
9      if (Enum.TryParse(str, out result))
10     {
11         Console.WriteLine(result);
12     }
13     else
14     {
15         Console.WriteLine("Parse error");
16     }
17 }
```

LinqPad

THANK YOU FOR YOUR ATTENTION!

Questions?

PROBLEM: NEW INTERFACE

```
4  interface IInterface
5  {
6      void Message();
7  }
8
9  void Main()
10 {
11     var foo = new IInterface();
12     foo.Message();
13 }
```

LinqPad

PROBLEM: BOOL VS BOOL

```
1  void Main()
2  {
3      bool bool1 = BoolStore.Bool1;
4      bool bool2 = BoolStore.Bool2;
5
6      Console.WriteLine(bool1);
7      Console.WriteLine(bool2);
8      Console.WriteLine($"{bool1} == {bool2} = ({bool1 == bool2})");
9  }
```

LinqPad

THE END

WHY DO C++ DEVELOPER NEED GLASSES? BECAUSE THEY DON'T C#