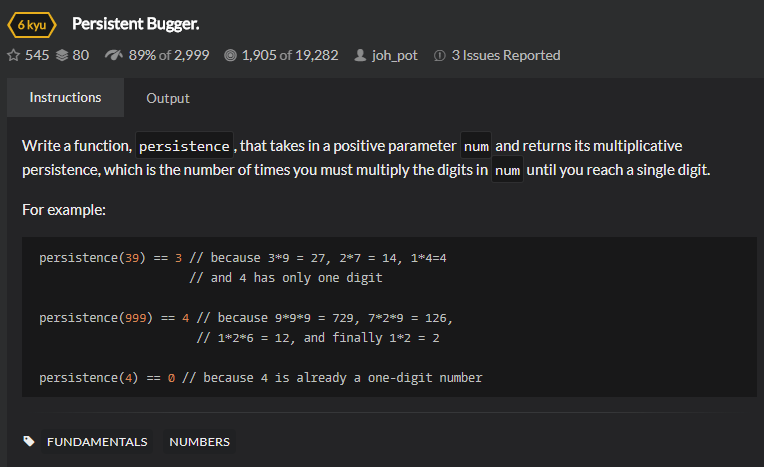
Day20. Hi,今天開始都是6kyu啦!Codewars\_Persistent Bugger

Codewars我回來惹，我好想你喔

LeetCode一直欺負我(被揍

今天開始應該都會寫6kyu的題目了

廢話我就不多說惹，今天的題目長這樣



題目跟前幾天寫的某提LeetCode的需求有點像

一樣要拆字元，一樣要運算，不過這題簡單一點，只要計算中間的運算幾次即可。

現在就來拆一下題目吧。

1. 將數字拆解->toString方法
2. 將拆解的數字相乘
3. 運算相乘次數

一開始我們先來寫一個輸入一個位元的測試吧

那就是從輸入0的開始吧!!

```

[TestMethod]

public void Input\_0\_Should\_Be\_0()

{

Assert.AreEqual(0, Persist.Persistence(0));

}

```

而Production Code 也就是老樣子會長成這個樣子

```

public static int Persistence(int n)

{

throw new System.**NotImplementedException**();

}  
```

老樣子，跑個測試，沒過很正常，紅燈，commit一下

接下來把Production Code改一下，用最簡單的方式解決他!

```

public static int Persistence(int n)

{

return 0;

}  
```

接下來跑個測試，PASS! Commit~

接下來寫一個輸入0過了，就要考慮到相乘的需求囉，所以要寫一個方法完成他。

在寫方法前就要寫了幾個測試。

```

[TestMethod]

public void DigitMutiply\_Input\_1\_Should\_Be\_1()

[TestMethod]

public void DigitMutiply\_Input\_11\_Should\_Be\_1()

[TestMethod]

public void DigitMutiply\_Input\_23\_Should\_Be\_6()

[TestMethod]

public void DigitMutiply\_Input\_123\_Should\_Be\_6()

```

所以現在要開始完成1(拆)、2(相乘)的需求囉!

而相乘的方法長這樣

```

public static int DigitMutiply(long n)

{

var result = Convert.ToInt32(n.ToString()[0].ToString());

for (int i = 1; i < n.ToString().Length; i++)

{

result \*= Convert.ToInt32(n.ToString()[i].ToString());

}

return result;

}

```

再來寫2個位元的Production Code測試吧!

```

[TestMethod]

public void Input\_11\_Should\_Be\_1()

{

Assert.AreEqual(1, Persist.Persistence(11));

}

```

跑個測試，Fail，Commit!

現在就要加入剛才的digitMutiply的方法到Production code裡，並且運算相乘的次數有多少個囉!

```

public static int Persistence(long n)

{

var result = 0;

while (n > 9)

{

n = DigitMutiply(n);

result++;

}

return result;

}

```

接下來跑一下測試吧，綠燈! Commit~

基本上目前的Code已經涵蓋到所有輸入了。

所以我們來補上幾個測試案例吧!

```

[TestMethod]

public void Input\_28\_Should\_Be\_2()

[TestMethod]

public void Input\_39\_Should\_Be\_3()

[TestMethod]

public void Input\_999\_Should\_Be\_4()

[TestMethod]

public void Input\_25\_Should\_Be\_2()

```

改完不要忘記跑測試，Pass，Commit!

再來仔細看一下會發現其實我們的Production Code是可以用遞迴完成的，就變成這個樣子

```

public static int Persistence(long n)

{

return n < 10 ? 0 : Persistence(DigitMutiply(n)) + 1;

}

```

接下來就可以提交到Codewars中了。

這是最後的Production Code

```

public class Solution

{

public static bool HasDigitOne(int i)

{

return i.ToString().Contains("1");

}

public static int CountDigitOne(int n)

{

return n > 0 ? Enumerable.Range(1, n).Count(HasDigitOne) : 0;

}

}

```

以下是今天所有的測試案例

```

[TestClass]

public class UnitTest1

{

[TestMethod]

public void Input\_0\_Should\_Be\_0()

{

Assert.AreEqual(0, Persist.Persistence(0));

}

[TestMethod]

public void Mutiply\_Input\_0\_Should\_Be\_0()

{

Assert.AreEqual(0, Persist.DigitMutiply(0));

}

[TestMethod]

public void DigitMutiply\_Input\_1\_Should\_Be\_1()

{

Assert.AreEqual(1, Persist.DigitMutiply(1));

}

[TestMethod]

public void DigitMutiply\_Input\_11\_Should\_Be\_1()

{

Assert.AreEqual(1, Persist.DigitMutiply(11));

}

[TestMethod]

public void DigitMutiply\_Input\_23\_Should\_Be\_6()

{

Assert.AreEqual(6, Persist.DigitMutiply(23));

}

[TestMethod]

public void DigitMutiply\_Input\_123\_Should\_Be\_6()

{

Assert.AreEqual(6, Persist.DigitMutiply(123));

}

[TestMethod]

public void Input\_11\_Should\_Be\_1()

{

Assert.AreEqual(1, Persist.Persistence(11));

}

[TestMethod]

public void Input\_28\_Should\_Be\_2()

{

Assert.AreEqual(2, Persist.Persistence(28));

}

[TestMethod]

public void Input\_39\_Should\_Be\_3()

{

Assert.AreEqual(3, Persist.Persistence(39));

}

[TestMethod]

public void Input\_999\_Should\_Be\_4()

{

Assert.AreEqual(4, Persist.Persistence(999));

}

[TestMethod]

public void Input\_25\_Should\_Be\_2()

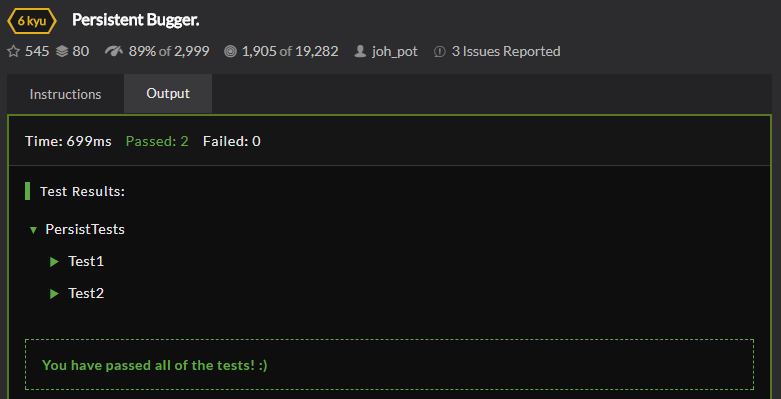
{

Assert.AreEqual(2, Persist.Persistence(25));

}

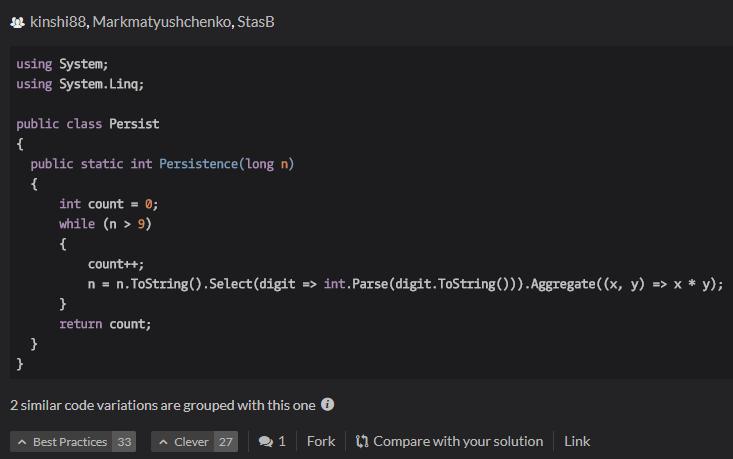
}

```



在Codewars上成功提交了~

來看一下其他人怎麼寫吧!



哦…..Aggregate的寫法啊…..

不會用XD

看來可以找時間好好理解這個東西了

不過其他人的寫法大致上跟我差不多就是惹

Git url :

<https://github.com/SQZ777/Codewars_PersistentBugger>

Codewars Link:

<https://www.codewars.com/kata/persistent-bugger/train/csharp>

下一題，明天見!