Week # 1 Machine Problems

Instructions:

1. Fork this Github repository: https://github.com/neilvillareal/DataStructureActivities
2. Clone the forked repository to your local machine.
3. Open the solution in Visual Studio
4. Complete or implement the requirements for the following functions specified below. ***Digits*** class can be found inside the Program.cs source file
5. Use the main method of the program to invoke the functions and evaluate the output. Or else, run the test cases in the Test Explorer of the attached test project.
6. Once completed, commit and push your code to your Github repo on or before Thursday (September 16,2021 12nn).
7. For inquiries, send it to my email: neilvillareal@gmail.com

1. Complete the function **AddDigits(n)** which accepts an integer and returns the sum of all its digits. For example:

> ***n*** = 29, the output should be ***AddDigits(n) = 11***.
> ***n*** = 123, the output should be ***AddDigits(n) = 6***.

2. Complete the function **LargestNumber(n)** which returns the largest number that contains exactly ***n*** digits. For example:

> ***n*** = 2, the output should be ***LargestNumber(n) = 99***
> ***n*** = 9, the output should be ***LargestNumber(n) = 9999***

3. Complete the function **OptimalEqualSplit(n,m)**. Assuming ***n*** children have got ***m*** pieces of candy. They want to eat as much candy as they can, but each child must eat exactly the same amount of candy as any other child. Determine how many pieces of candy will be eaten by all the children together. Individual pieces of candy cannot be split. For example:

> ***n*** = 3 and ***m*** = 10, the output should be **OptimalEqualSplit(n, m)** = 9
> ***n*** = 10 and ***m*** = 5, the output should be **OptimalEqualSplit(n, m)** = 0
> ***n*** = 4 and ***m*** = 15, the output should be ***OptimalEqualSplit(n, m)*** = 12

4. Complete the function **Persistence(n)** which takes in a positive parameter ***n*** and returns its multiplicative persistence, which is the number of times you must multiply the digits in ***n*** until ***n*** reaches a single digit. For example:

n = 39, the output should be **Persistence(n)** == 3
// because 3*9 = 27, 2*7 = 14, 1*4=4 and 4 has only one digit

n = 999, the output should be **Persistence(n)** == 4
// because 9*9*9 = 729, 7*2*9 = 126, 1*2*6 = 12, and finally 1*2 = 2

n = 4, the output should be **Persistence(n)** == 0
// because 4 is already a one-digit number

NOTE: do not use or include any namespaces such as System.Linq, System.Collections.Generic, etc.