F# Tutorial

Pipe-Forward Operator

February 23, 2018

1 Modified Project Euler Solutions

`IsPrime` Function Provided

The following function determines whether a positive integer `x` is a prime number or not. It is already provided, and we do not need to re-implment it.

```
let IsPrime x =
    let squareRoot = x |> double |> sqrt |> int
    if x = 1 then false
    else if x = 2 then true
    else if x % 2 = 0 then false
    else
        [3 .. 2 .. squareRoot]
        |> List.forall (fun i -> x%i <> 0)

// val IsPrime: x:int -> bool
```

Question 1

`https://projecteuler.net/problem=1`

Original Question. *Implement a function that sums up all multiples of* 3 *or* 5 *in a list.*

```
let SumMultiplesOf3Or5 xList =
    xList
    |> List.filter (fun x -> x % 3 = 0 || x % 5 = 0)
    |> List.sum
```

Question 2

Original Question. *The Fibonacci sequence (starting with* 1 *and* 2*) looks something like:*

$$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

(For example, $1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8$*, etc.)*
Find the sum of all even-valued fibonacci numbers below 4 *million.*

1. We will first test whether the 41st fibonacci number exceeds four million or not.

```
let first40FibNumbers =
    [1 .. 40]
    |> List.scan (fun (x,y) _ -> (y, x + y)) (1,2)
// Result: [(1,2); (2,3); ......; (267914296, 433494437)]
```

   And so, we only need to consider the first 40 or 41 Fibonacci number. (In fact, we do not even need to consider beyond the 32th number)

2. Sum all even-valued fibonacci numbers below 4 million.

```
let fibSum =
    [1 .. 40]
    |> List.scan (fun (x,y) _ -> (y, x + y)) (1,2)
    |> List.map (fun (x,y) -> x)
    |> List.filter (fun x -> x % 2 = 0)
    |> List.filter (fun x -> x < 4000000)
    |> List.sum
// Result: 4613732
```

Question 3

Exercise (Euler Project Question 3)

`https://projecteuler.net/problem=3`
Please see the next section, where we approach the original Question 3.

Modified Question. *Write a function that takes a list of (positive) integers, and returns the largest prime number in that list.*

```
let FindLargestPrime intList =
    intList
    |> List.filter (IsPrime)
    |> List.max
```

Question 4

`https://projecteuler.net/problem=4`

Original Question. *A palindromic number reads the same from left-to-right or right-to-left.*

The largest palindromic number made from the product of two 2-digit numbers is $9009 = 91 \times 99$.

Find the largest palindrome made from the product of two 3-digit numbers.

The following `IsPalindrome` function that is already implemented. You do not need to re-implement it.

```
let ReverseString (xString: string) =
    new string (xString.ToCharArray() |> Array.rev)

let IsPalindrome xString =
    (ReverseString xString) = xString

let palindromeResult1 = IsPalindrome "ASDF"
let palindromeResult2 = IsPalindrome "ABCCBA"
// val palindromeResult1 : bool = false
// val palindromeResult2 : bool = true
```

Find the largest palindrome number which is a product of two 3-digit numbers $a \times b$, where $100 \leq a \leq 999$, and $100 \leq b \leq 999$

```
let findProductPalindrome =
    List.allPairs [100 .. 999] [100 .. 999]
    |> List.map (fun (a,b) -> a * b)
    |> List.filter (fun product ->
        product
        |> string
        |> IsPalindrome
    )
```

Question 5

`https://projecteuler.net/problem=5`

Modified Question. *Given a list of integers, find the lowest common multiple (LCM) of all those numbers. (Assume no integer overflow)*

Remark: We are given the following GCD and LCM functions. We do not need to re-implement them.

```
let rec gcd x y =
    if x < 0 || y < 0 then failwith "cannot accept negative
    numbers"
    if x > y then gcd y x
    else if x = 0 then y
    else gcd (y % x) x

let lcm a b =
    a * b / (gcd a b)
```

Solution:

```
let lcmOfList xList =
    xList
    |> List.fold lcm 1

let result11 = lcmOfList [1 .. 10]
// Result: 2520

let result12 = lcmOfList [2;3;4;6;8;12]
// Result: 24
```

Remark: This method will FAIL for `xList = [1 .. 20]` because of integer overflow. Please see the next section on how to handle the original question.

Question 6

`https://projecteuler.net/problem=6`

Original Question. *Given a list of integers $x_1, x_2, \ldots, x_n$, write a function that calculates the following:*

$$\left(\sum_{i=1}^{n} x_i\right)^2 - \left(\sum_{i=1}^{n} {x_i}^2\right)$$

```
let ProjectEulerProblem6 xList =
    let sumOfSquares =
        xList
        |> List.map (fun x -> x * x)
        |> List.sum

    let sum =
        xList
        |> List.sum

    // return
    (sum * sum) - sumOfSquares
```

```
let result13 = ProjectEulerProblem6 [1 .. 100]
```

Question 7

`https://projecteuler.net/problem=7`

Original Question. *The list of prime numbers are* $2, 3, 5, 7, 11, 13, \ldots$. *We can see that the* 6*th prime number is* 13.

What is the 10001*th prime number?*

We start with a random guess of 500000:

1. **Solution part (a)**: How many prime numbers are there between 2 and 500000?

```
let numberOfPrimesWithinRange =
    [2 .. 500000]
    |> List.filter (IsPrime)
    |> List.length
```

Expected answer: 41538.

2. **Solution part (b)**: What is the 10001th prime number between 2 and 500000?

Because of 0-based indexing, we use `(List.item 10000)` to find the 10001th prime number (which is between 2 to 500000).

```fsharp
let find10001thPrime =
    [2 .. 500000]
    |> List.filter (IsPrime)
    |> List.item 10000
```

Question 8

https://projecteuler.net/problem=8

Modified Question. *Given a list of digits, find four adjacent digits with the largest product. For example, in the following number:*

$$731671765313306249192251\mathbf{9674}4265747423553491949 34$$

The 4 consecutive digits that gives the largest product is $9 \times 6 \times 7 \times 4 = 9674$
(Notice that this line is the first line in the original question)

```fsharp
let digitList =
    [7;3;1;6;7;1;7;6;5;3;1;3;3;0;6;2;4;9;1;.....]

let result8 =
    digitList
    |> List.windowed 4
    |> List.map (fun x -> x, ListProduct x)
    |> List.maxBy (fun (_,product) -> product)
// val result8 : int list * int = ([9;6;7;4], 1512)
```

Please see the next section on how we approach the original question.

Question 9

https://projecteuler.net/problem=9

Original Question. *Find the only Pythagorean triplet a, b, c that satisfy:*

$$a < b < c, \qquad a + b + c = 1000, \qquad a^2 + b^2 = c^2$$

```
let FindPythagoreanTriple =
    List.allPairs [1 .. 1000] [1 .. 1000]
    |> List.filter (fun (a,b) ->
        let c = 1000 - a - b
        a * a + b * b = c * c
    )
// Result: [(200, 375); (375, 200)]
```

Question 10

Exercise (Euler Project Question 10)

https://projecteuler.net/problem=10

Please see the next section where we work with large numbers (and potential integer overflow)

Modified Question. *Given a number $N < 200,000$, find the sum of all prime numbers between 2 and N.*

```
let TotalSumOfPrimeLessThan N =
    [2 .. N]
    |> List.filter (IsPrime)
    |> List.sum
```

2 Original Project Euler Solutions

Question 1

We did not modify Question 1.

Question 2

We did not modify Question 2.

Question 3

PLEASE EDIT THIS.

Question 4

We did not modify Question 4.

Question 5

PLEASE EDIT THIS.

Question 6

We did not modify Question 6.

Question 7

We did not modify Question 7.

Question 8

PLEASE EDIT THIS.

Question 9

We did not modify Question 9.

Question 10

`https://projecteuler.net/problem=10`

Original Question. *The sum of the primes below* 10 *is* $2 + 3 + 5 + 7 = 17$
Find the sum of all the primes below two million $(2,000,000)$.

```fsharp
open System.Numerics

let Version2_TotalSumOfPrimeLessThan N =
    [2 .. N]
    |> List.filter (IsPrime)
    |> List.map (BigInteger)
    |> List.sum

// Remark: The code below can take 10 seconds, as this is not the most optimal algorithm.
let result17 = Version2_TotalSumOfPrimeLessThan 2000000
// Result: 142913828922
```

3 Fold, Scan, State

Key concept:

1. The fold and scan functions are used to keep track of states.
    - It can be considered as eliminating a lot of intermediate steps, where the number of intermediate steps may change based on the length of the list.
    - It is somewhat similar to using a mutable state, but less things to keep track of.

3.1 `List.fold`

Let us look at an example:

```
type Commands =
    | TIMES2
    | ADD1

let listOfCommands1 =
    [TIMES2; ADD1; TIMES2; TIMES2]
```

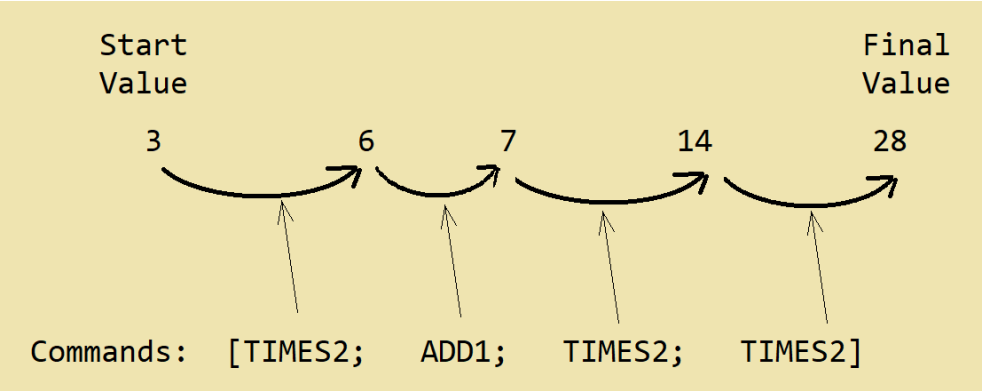
Here, `Commands` is a discriminated union (i.e. an abstract data type) that only has two different possible values. So, it is a little bit safer than using a list of strings (because strings can take on a lot of values)

```
let ChangingFunction prevResult currentCommand =
    match currentCommand with
    | TIMES2 -> prevResult * 2
    | ADD1 -> prevResult + 1

let startingValue = 3
```

Here, `ChangingFunction` tells us how to modify a value based on the `Commands` accepted.

```
let result1 =
    List.fold ChangingFunction startValue listOfCommands1
// val result1 : int = 28
```



Here, we have a `startValue` of 3, and we go through the `listOfCommands1` and evolve the `startValue` based on the `ChangingFunction`.

An equivalent implementation would be the following:

```
let result1_version2 =
    let intermediateResult1 =
        ChangingFunction startValue listOfCommands1.[0]

    let intermediateResult2 =
        ChangingFunction intermediateResult1
            listOfCommands1.[1]

    let intermediateResult3 =
        ChangingFunction intermediateResult2
            listOfCommands1.[2]

    let finalResult =
        ChangingFunction intermediateResult3
            listOfCommands1.[3]

    finalResult
```

Or if we use mutable, then:

```
let result1_version3 =
    let mutable valueSoFar = startValue
    for command in listOfCommands1 do
        let updatedValue =
            ChangingFunction valueSoFar command
        valueSoFar <- updatedValue
    // return
    valueSoFar
```

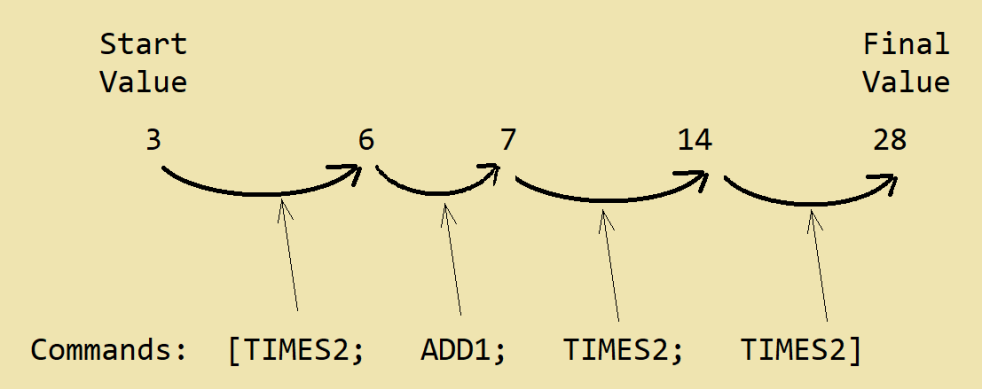
This is most similar to codes that you may write in Java/C++

Warning: If you use VisualStudio /VisualStudioCode, you may see that `valueSoFar` is highlighted yellow in your editor, as a warning that there is a mutable value in our program. As mentioned before, F# discourages the usage of mutable values.

3.2 Dependance on Starting and Intermediate Value

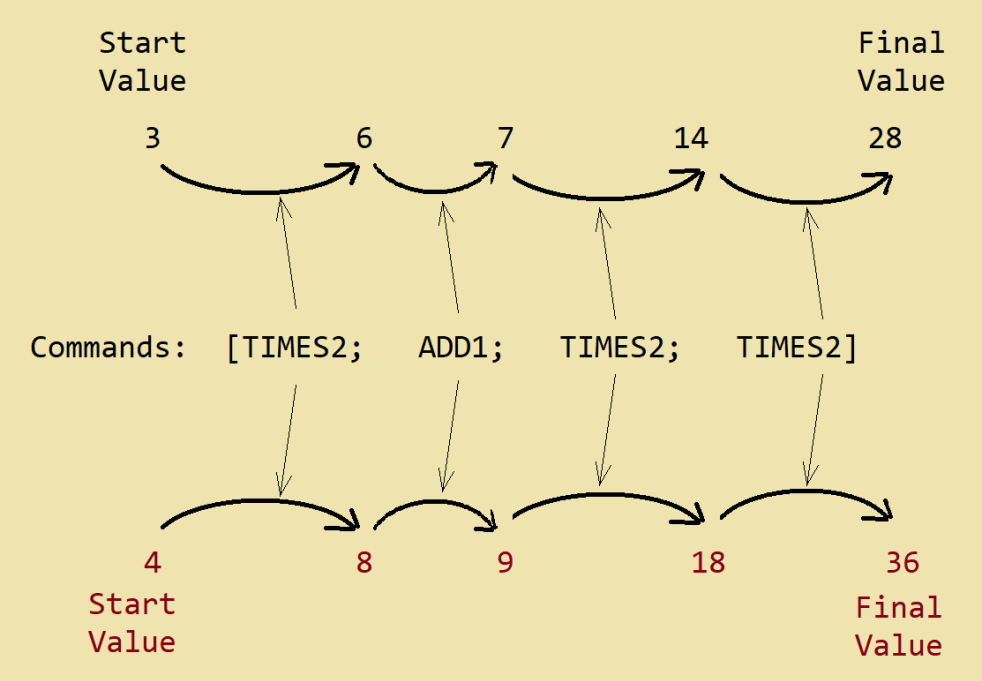
Dependance on Starting Value

```
let result1 =
    List.fold ChangingFunction startValue listOfCommands1
// val result1 : int = 28
```



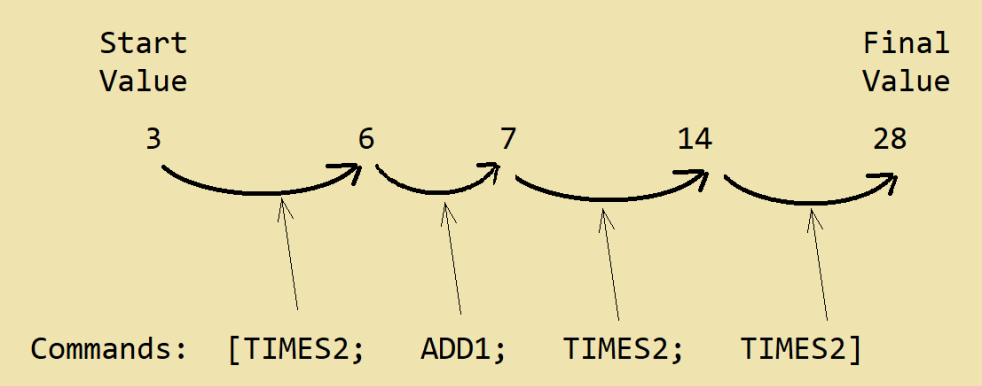
Notice that the folding process depends on the starting value:

```
let startValue2 = 4
let result2 =
    List.fold ChangingFunction startValue2 listOfCommands1
// val result2 : int = 36
```



Dependance on Intermediate Value

```
let result1 =
    List.fold ChangingFunction startValue listOfCommands1
// val result1 : int = 28
```

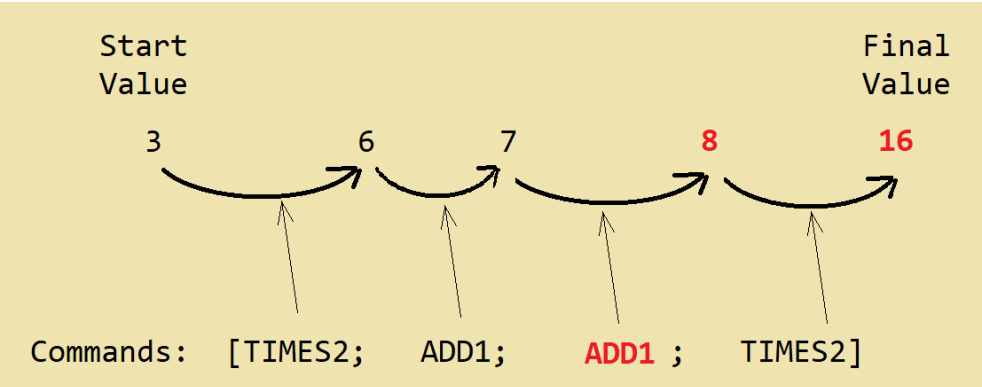


The folding process also depends on the intermediate values:

```
let listOfCommands2 =
    [TIMES2; ADD1; ADD1; TIMES2]

// startValue = 3

let result3 =
    List.fold ChangingFunction startValue listOfCommands2
```



3.3 Examples

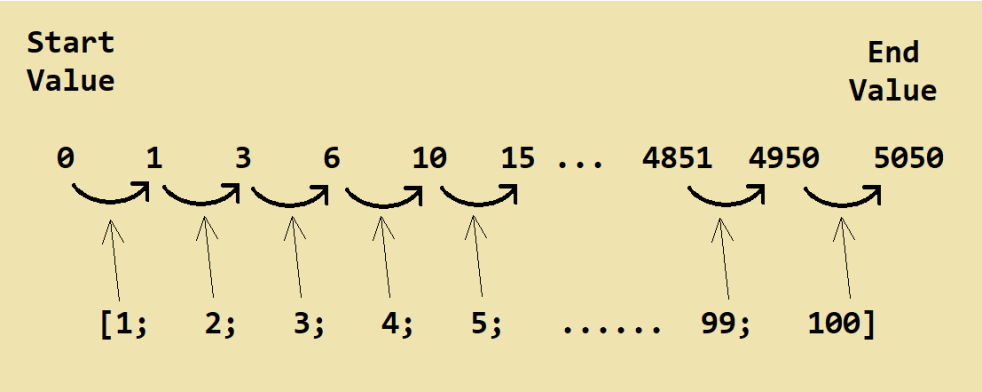
Sum a List

In order to sum a list, we can just use `List.sum`

```
let result4 = List.sum [1 .. 100]
// val result4 : int = 5050
```

Alternatively, we can imagine that we are adding up the value one by one, with a starting value of 0.

```
let result5 =
    [1 .. 100]
    |> List.fold (fun acc y -> acc + y) 0
// val result5 : int = 5050
```



Notice that the "folding function" is an anonymous/lambda function that accepts two variables:

- `acc`: The "accumulator" or intermediate result that is used to accumulate the informations.
- `y`: The elements from the list.

And the result of the "folding function" is the updated "accumulator" that will be passed on to the next accumulation stage.

If we are working in the previous case (where the accumulator is an integer, and the element of the list is a discriminated union `ADD1, TIMES2`), then it is very clear which variable is which in the anonymous function.

However, in this case of re-implementing `List.sum`, both the accumulator and the element of the list are `int`, and so we may need to be careful when we are using a non-symmetric operator, i.e:

$$a + b = b + a \qquad a - b \neq b - a$$

Product of a List

Question. *Write a function that takes in an integer list, and outputs the product of all elements in that list (assume no integer overflow).*

When you use `List.fold`, consider two things:

1. Which starting value should you use?
2. What does your accumulator function do?

```
let ListProduct xList =
    xList
    |> List.fold (fun acc y -> ......)  .......

// What accumulating/folding function do you want to use?
// Which starting value should you use?
```

```
let result6 = ListProduct [1 .. 5]
// Expected Result: 1 x 2 x 3 x 4 x 5 = 120

let result7 = ListProduct [2; 3; 5; 7; 11; 13]
// Expected Result: 2 x 3 x 5 x 7 x 11 x 13 = 30030
```

Application: Euler Project Question 8

`https://projecteuler.net/problem=8`

Modified Question. *Given a list of digits, find four adjacent digits with the largest product. For example, in the following number:*

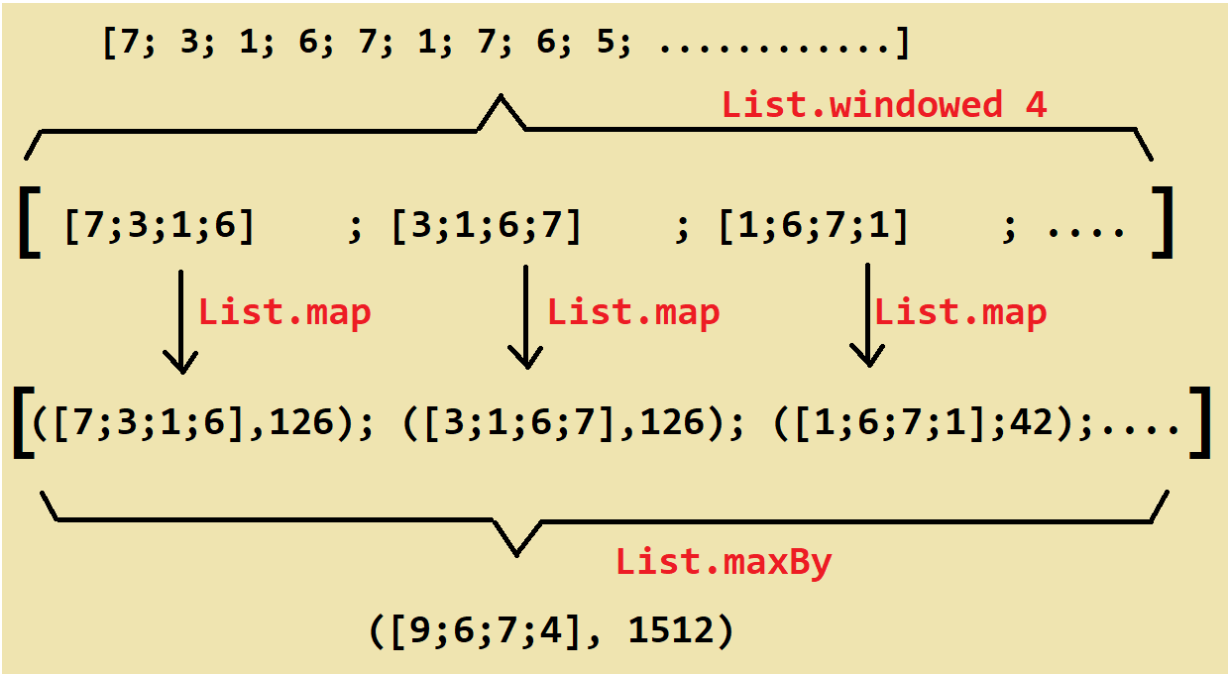
$$731671765313306249192251\mathbf{9674}4265747423553491949 34$$

The 4 consecutive digits that gives the largest product is $9 \times 6 \times 7 \times 4 = 9674$
(Notice that this line is the first line in the original question)

```
let digitList =
    [7;3;1;6;7;1;7;6;5;3;1;3;3;0;6;2;4;9;1;.....]

let result8 =
    digitList
    |> List.windowed 4
    |> List.map (fun x -> x, ListProduct x)
    |> List.maxBy (fun (_,product) -> product)
// val result8 : int list * int = ([9;6;7;4], 1512)
```

We will use the picture below to illustrate what we are trying to achieve here.



Our goal here is to show that the `ListProduct` that we have implemented before using `List.fold` can be very powerful when combined with other `List` functions (e.g. `List.windowed`, `List.maxBy`, etc.) . To see how we approach the original Euler Problem, see the appendix.

Example: GCD of a list of integers

GCD for two variables already provided

You are given the following recursive `rec` function, that helps to calculate the greatest common divisor (GCD) of two integers. (This is Euclidean Algorithm)

```
let rec gcd x y =
    if x < 0 || y < 0 then failwith "cannot accept negative
    numbers"
    if x > y then gcd y x
    else if x = 0 then y
    else gcd (y % x) x
```

Reminder: You do not need to re-implement this function. You can just use it.

Question. *Given a list of (positive) integers, find the greatest common divisor (GCD) of those integers.*

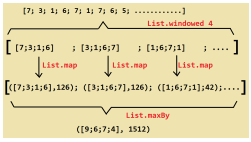
Strategy: We make the following observation:

$$\begin{aligned} GCD\left(x_1, x_2, x_3\right) &= GCD\left[GCD\left(x_1, x_2\right), x_3\right] \\ GCD\left(x_1, x_2, x_3, x_4\right) &= GCD\left[GCD\left(x_1, x_2, x_3\right), x_4\right] \\ \vdots \qquad &= \qquad \vdots \\ GCD\left(x_1, x_2, \ldots, x_n\right) &= GCD\left[GCD\left(x_1, \ldots, x_{n-1}\right), x_n\right] \end{aligned}$$

```
let gcdOfList xList =
    let first = xList |> List.head
    let remaining = xList |> List.tail

    remaining
    |> List.fold gcd first
```

We will use the picture below to illustrate what we are trying to achieve here.



Exercise: Euler Project Question 5

`https://projecteuler.net/problem=5`

Modified Question. *Given a list of integers, find the lowest common multiple (LCM) of all those numbers. (Assume no integer overflow)*

Strategy: We make the following observation:

$$\begin{aligned}
LCM\left(x_1, x_2\right) &= LCM\left[LCM\left(x_1\right), x_2\right] \\
LCM\left(x_1, x_2, x_3\right) &= LCM\left[LCM\left(x_1, x_2\right), x_3\right] \\
LCM\left(x_1, x_2, x_3, x_4\right) &= LCM\left[LCM\left(x_1, x_2, x_3\right), x_4\right] \\
\vdots \qquad &= \qquad \vdots \\
LCM\left(x_1, x_2, \ldots, x_n\right) &= LCM\left[LCM\left(x_1, \ldots, x_{n-1}\right), x_n\right]
\end{aligned}$$

Hint: You can directly use the LCM function as your folding function. You do not need to re-implement it.

```
let lcm a b =
    a * b / (gcd a b)
```

So, we do not need to worry about our folding function, and we just need to worry about the starting value.

```
let lcmOfList xList =
    xList
    |> List.fold lcm ........

let result11 = lcmOfList [1 .. 10]
// Result: 2520

let result12 = lcmOfList [2;3;4;6;8;12]
// Result: 24
```

Warning: If you try to use this function on the list `[1 .. 20]`, you may either see an error, or see the following wrong result:

```
let result11 = lcmOfList [1 .. 20]
// Wrong Result: 18044195
// WRONG RESULT!!!!!!
```

Again, this is because of integer overflow (`int` cannot handle large numbers). You can see the Appendix on how to handle this situation.

Exercise: Euler Project Question 2

A Appendix

A.1 Project Euler In-Depth

A.1.1 Euler Project Question 3

`https://projecteuler.net/problem=3`

In the main chapter, we asked you to solve a modified version of Project Euler Q3:

Modified Question. *Write a function that takes a list of (positive) integers, and returns the largest prime number in that list.*

Hopefully, by using the `IsPrime` function provided in the main chapter, your answer is:

```
let FindLargestPrime intList =
    intList
    |> List.filter (IsPrime)
    |> List.max
```

How is this related to the original question?

Question. *Given an integer Z, write a function that finds the largest prime factor of Z. e.g. The prime factors of* 13195 *are* $5, 7, 13, 29$*, and so the largest for* 13195 *is* 29*.*

Problem Analysis

In our "modified approach" in the original text, we tried to find the largest prime factor of Z between 2 and $\sqrt{Z}$. However, given an integer Z, it is possible that the largest prime factor of Z is greater than $\sqrt{Z}$

- Example: $3 \times 7 = 21$. The largest prime factor is $7 > \sqrt{21} \approx 4.58$.
- Example: $6 \times 11 = 66$. The largest prime factor is $11 > \sqrt{66} \approx 8.12$.

We will modify our approach to the following method:

1. Let $S_1 = \{a_1, \ldots, a_n\}$ be all the factors of Z (not necessarily prime factors) between 1 and $\sqrt{Z}$. This set will always contain at least one element: $a_1 = 1$.
2. Let $S_2 = \left\{\frac{Z}{a_1}, \ldots, \frac{Z}{a_n}\right\}$. These are all the factors of Z between $\sqrt{Z}$ and Z. This set will always contain at least one element: $\frac{Z}{a_1} = Z$.
3. So, $S_1 \cup S_2 = \left\{a_1, \ldots, a_n, \frac{Z}{a_1}, \ldots, \frac{Z}{a_n}\right\}$ are all the factor of Z (not necessarily prime factors).
4. Out of our list of candidates $S_1 \cup S_2$, which number is the largest, prime number?

Working with `BigInteger`:

1. We will need an `IsPrimeBigInteger` function that helps us check whether a `BigInteger` is a prime number or not.

```
let IsPrimeBigInteger x =
    let squareRoot = x |> double |> sqrt |> BigInteger
    if x = BigInteger(1) then false
    else if x = BigInteger(2) then true
    else if x % BigInteger(2) = BigInteger(0) then false
    else
        [BigInteger(3) .. BigInteger(2) .. squareRoot]
        |> List.forall (fun i -> x%i <> BigInteger(0))
```

2. When `x, y` are both `BigIntegers`, then cannot do `x % y = 0`, because we cannot directly compare a `BigInteger` with an integer `0`. We need to do:

```
x % y = BigInteger(0)
```

Code Solution

```
open System.Numerics

let FindLargestPrimeFactor (Z: BigInteger) =
    let approxSqrt = Z |> double |> sqrt |> BigInteger

    // Find factors of Z between [2 .. sqrt(Z)]
    // Not necessarily prime factors.
    let list1 =
        [BigInteger(2) .. approxSqrt]
        |> List.filter (fun x -> Z % x = BigInteger(0))

    // Produce another list such that:
    // For each element "a" in list1, it gives "Z / a"
    let list2 =
        list1
        |> List.map (fun a -> Z / a)

    // List.append combines the two lists.
    let combinedList =
        List.append list1 list2
    // Choose only prime numbers from the combinedList, and
    find the maximum using List.max
    combinedList
    |> List.filter (IsPrimeBigInteger)
    |> List.max
```

Test:

```
let number1 = BigInteger(21)
let result18 = FindLargestPrimeFactor number1
// Expect result: 7

let number2 = BigInteger(66)
let result19 = FindLargestPrimeFactor number2
// Expect result: 11

let number3 = BigInteger.Parse("600851475143")
let result20 = FindLargestPrimeFactor number3
// Expect result: 6857
```

Notice that the provided solution for the original question has the following last two lines:

```
    ....................
    |> List.filter (IsPrimeBigInteger)
    |> List.max
```

On the other hand, the solution for the modified question has the following last two lines:

```
    ....................
    |> List.filter (IsPrime)
    |> List.max
```

And so, the original question is all about:

1. Finding a list of candidates (that is based on mathematics consideration, and less about computing/programming)

2. From the list of candidates, extract out the largest prime number from the list.

So, we let the readers focus on the second part of the original problem in our original text. The trickiness of the original question is mainly mathematical, and we do not want that to distract you from the programming part of F#.

A.2 Optional Topics

A.2.1 inline functions

On some occasion, if you need to use the same function on different type which supports `(*)`, then you can use the `inline` keyword.

```
let inline Product x y = x * y

let multiply2Int = Product 2 3
printfn "Multiply the two numbers gives: %i" multiply2Int
// Output: "Multiply the two numbers gives: 6"

let multiply2Double = Product 3.0 4.0
printfn "Multiply the two numbers gives: %f"
   multiply2Double
// Output: "Multiply the two numbers gives: 12.000000"
```

However, not every datatype supports multiplication `(*)`

```
let multiply2WordsError = Product "word1" "word2"
"ERROR!!!!!!!!!!"
```

INPUT PICTURE HERE!

Similarly, we can do this:

```
let inline CustomAdd x y z = x + y + z
let add3IntegerResult = CustomAdd 4 5 6
printfn "Adding the three integers give: %i"
   add3IntegerResult
// Output: "Adding the three integers give: 15"

let add3StringResult = CustomAdd "John " "F." " Kennedy"
printfn "Concatenate the three strings give: %s"
   add3StringResult
// Output:
// "Concatenate the three strings give: John F. Kennedy"

let add3DecimalResult = CustomAdd 10.3 10.2 10.1
printfn "Adding the three decimals give: %f"
   add3DecimalResult
// Output: "Adding the three decimals give: 30.600000"
```

However, not every datatype supports addition `(+)`

```
let add3BooleanError = CustomAdd true false false
"ERROR!!!!!!!!!!"
```

INPUT PICTURE HERE!