3 List in F#

Key Concept:

1. Introduce basic `List` functions
   (a) `List.filter`
   (b) `List.map`
2. Code in F# are very easy to understand (thanks for pipe-forward operator `|>` and the F# language design)
3. Anonymous functions / lambda function also helps.
   - You are defining a function at the exact location where it is most useful. So it boosts productivity.
   - `fun` is a keyword in F#!

3.1 Creating a list

You can create a list of integers/ float / string using the following notations:

```
let list1 = [1 .. 100]
let list2 = [50 .. 80]
let list3 = [1 .. 2 .. 100]

let list4 = [1.0 .. 100.0]
let list5 = [0.0 .. 0.05 .. 1.0]

let list6 = [1; 20; 50; 100; 55; 5; 10]
let list7 = [1.0; 6.0; 5.0; 10.0; 3.0; 2.0]

let list8 = ["ABC";"DEF";"GHI";"JKL";"MNO"]
```

The `;` is used to separate different elements, and `[a .. b]`, `[a .. diff .. b]` is used to specify any increasing/decreasing pattern.

If you hover your mouse on top of those variables (using VisualStudio or VisualStudioCode), you will see the types are `int list`, `float list`, `etc`. An alternate notation would be `List<int>`, `List<double>`, `etc`.

Warning: You cannot create a list with different types, e.g. the example below tries to create a list with a string, an integer, and a decimal/float.

```
let listError = ["ABC"; 123; 400.0]
// ERROR! Cannot define different type in the same list!
```

3.2 `List.filter`

Here is a simple function that returns true/false, depending on whether x is divisible by 2:

```
let IsItEven x = (x % 2 = 0)

let trueOrFalse1 = IsItEven 10
let trueOrFalse2 = IsItEven 3
```

Remark: `x % 2` means the remainder after we divide x by 2.

We can use this function together with `List.filter`:

```
let result1 = List.filter IsItEven [1 .. 100]
// Output:
// [2; 4; 6; ......; 98; 100]
```

The `List.filter` function filters a list, and only select the elements which satisfy some requirement; the requirement is specified through a function `IsItEven`.

Alternatively, because the definition of `IsItEven` is quite easy, we can even implement it immediately after `List.filter`, at the point where we need it the most.

```
let result2 = List.filter (fun x -> x % 2 = 0) [1 .. 100]
// Output:
// [2; 4; 6; ......; 98; 100]
```

The notation `(fun x -> x % 2 = 0)` is used to define anonymous/lambda function, i.e. functions that are easy to define, that we do not need to give it a name, e.g. `IsItEven`.

Benefits:

- We define this function using the `fun` keyword at exactly where it is used.
- If we define too many custom functions, e.g. `IsItEven`, then it will be hard to keep track when we have 1000+ functions, and we will lose productivity.

Remark: The code `(fun x -> x % 2 = 0)` represents a "thing", and that "thing" is a function, just like `IsItEven` is a function.

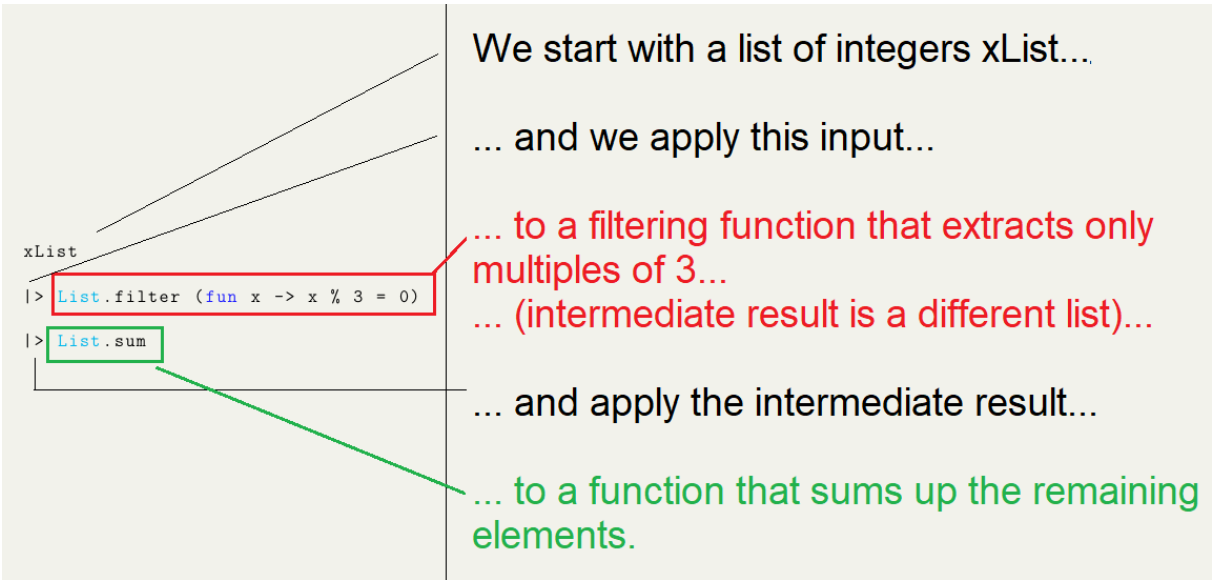
Side note: `fun` is a keyword in the F#! Programming in F# is very fun!

`List.filter` and Pipe-Forward `|>`

Let us look at the following function:

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

How to interpret this function:



So, F# is able to express all of these calculations with just 3 lines of code, which is quite elegant, maybe similar to Python code (in style), compared to other more traditional languages (Java/C++) which we need to write longer.

Using this function:

```
// 3 + 6 + 9 + ... + 99 = 1683
let result3 = SumMultiplesOfThree [1 .. 100]

// 3 + 6 + 9 + ... + 198 = 6633
let result4 = SumMultiplesOfThree [1 .. 200]
```

Output:

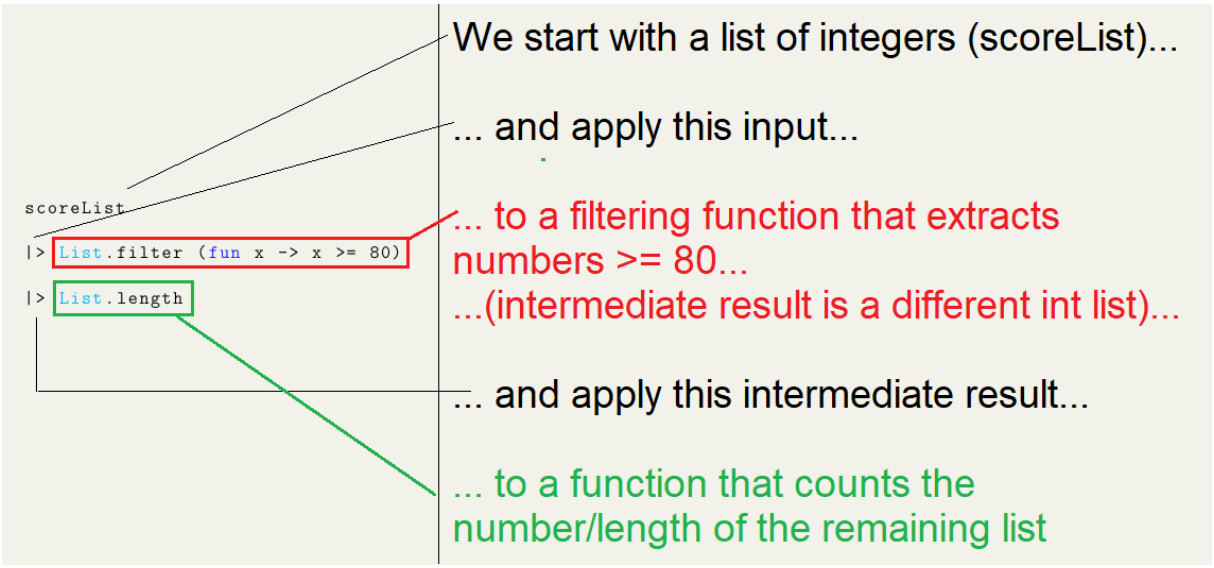
```
// val result3 : int = 1683
// val result4 : int = 6633
```

Another example

Let's say you want to find out how many students in your class got at least 80 points in an exam.

```
let CountGreaterThan80 scoreList =
    scoreList
    |> List.filter (fun x -> x >= 80)
    |> List.length
```

How to interpret this function:



Using this function:

```
let result5 =
    CountGreaterThan80 [60; 65; 70; 75; 80; 85; 90; 95]
printfn "%i students scored 80 or above." result5
```

Output:

```
// "4 students scored 80 or above."
```

Another example

This function adds up all multiples of 3, e.g. $3, 6, 9, \ldots$, but ignore all multiples of 5, e.g. $5, 10, \underline{\mathbf{15}}, 20, 25, \underline{\mathbf{30}}, \ldots$

```
let SumMultiplesOf3ButNot5 xList =
    xList
    |> List.filter (fun x -> (x % 3 = 0) && (x % 5 <> 0))
    |> List.sum

let result6 = SumMultiplesOf3ButNot5 [1 .. 100]
```

Output:

```
// val result6 : int = 1368
```

Remark:

- `(x % 3 = 0)`: is x divisible by 3?
- `(x % 5 <> 0)`: is x NOT a multiple of 5?

Exercise (Euler Project Question 1)

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

```
let SumMultiplesOf3Or5 xList =




    // Hint:  || means OR, && means AND

// From 1 to 10, the multiples of 3 or 5 are 3, 5, 6, 9, 10
// 3 + 5 + 6 + 9 + 10 = 33
let result7 = SumMultiplesOf3Or5 [1 .. 10]

// Expected answer: 233168
let result8 = SumMultiplesOf3Or5 [1 .. 999]
```

Remark: After you have completed this question, you can create an account and submit your solution here for personal achievement/accomplishment.

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

Exercise (Euler Project Question 7)

`IsPrime` Function Provided

You are given the following function that determines whether a positive integer `x` is a prime number or not. You can just directly use it. You do not need to implement it yourself.

```
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
```

Reminder: You can directly use the `IsPrime` function in the previous page. You do not need to re-implement it again.

`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 will solve this problem in two steps, starting with a random guess of 500000:

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

   Use the `IsPrime` function and the `List.length` function to determine how many prime numbers are between 2 and 500000. (Be careful: It is `List.length`, not `List.Length`)

   ```
   let numberOfPrimesWithinRange =
       [2 .. 500000]
       |> ......


       // Calculate how many primes are between 2 and 500000
       // Use "IsPrime", and later "List.length"
   ```

   Expected answer: 41538.

   This means that there are 41538 prime numbers between 2 and 500000, and so the 10001th prime number that we are looking for is also in this range (we could have chosen a smaller range, but 2 to 500000 is good enough).

2. The `List.item` function can be used to extract an item at an index/location. However, be careful that index/locations are 0-based. e.g.

```
let word1 = List.item 3 ["A"; "B"; "C"; "D"; "E"]
// val word1 : string = "D"

let word2 = List.item 5 ["A"; "B"; "C"; "D"; "E"; "F"; "G"
   ; "H"; "I"; "J"]
// val word2 : string = "F"
```

   So, to find the 10001th element of a list, you need to use `(List.item 10000)`.

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

   Use `IsPrime` function and `(List.item 10000)` to find the 10001th prime number (which is between 2 to 500000).

```
let find10001thPrime =
    [2 .. 500000]
    |> ......


    // Use "IsPrime", and later "List.item 10000"
```

   Expected answer: 104743.

Remark: After you have completed this question, you can create an account and submit your solution online for personal achievement/accomplishment.

Exercise (Euler Project Question 3)

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

Original Question. *The prime factors of* 13195 *are* $5, 7, 13, 29$.
What is the largest prime factor of the number 600851475143*?*

We will not attempt the original question. Instead, we will try a simpler problem:

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

Hint: You can reuse the `IsPrime` function from the previous question.

```
let FindLargestPrime intList =
    intList
    |> ......


    // Hint: Use "IsPrime", and later "List.max"
```

```
let primeResult1 = FindLargestPrime [2;3;5;7;11]
// Expected Result: 11

let primeResult2 = FindLargestPrime [7; 100; 200; 333; 777]
// Expected Result: 7
// Because only 7 is a prime number in this list

let primeResult3 = FindLargestPrime [100; 200; 300; 400;
   500]
// ERROR!
// Expected an error to occur, because there are no prime
   numbers,
// And so we cannot find the maximum of no numbers.
```

To see how the Modified Question 3 is related to the Original Question 3, please see the other document.

Exercise (Euler Project 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 $(2,000,000)$.

We will not attempt the original question. Instead, we will do this modified question:

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

Hint: Again, use the `IsPrime` function before. Do not re-implement the function.

```
let TotalSumOfPrimeLessThan N =
    [2 .. N]
    |>..................

    // Hint: Use "IsPrime". Do not re-implement it.
```

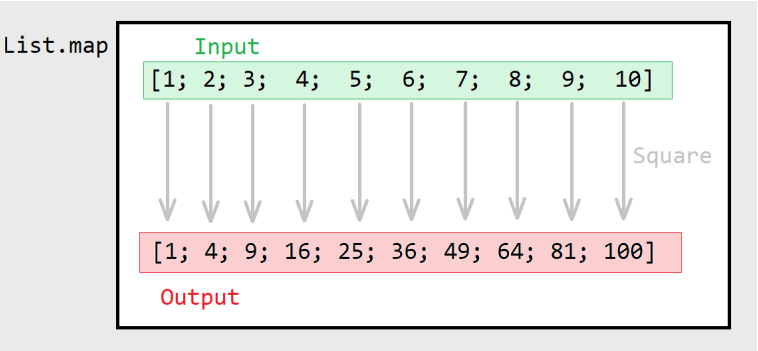
```
let primeSum1 = TotalSumOfPrimeLessThan 10
// 2 + 3 + 5 + 7 = 17

let primeSum2 = TotalSumOfPrimeLessThan 20
// 2 + 3 + 5 + 7 + 11 + 13 + 17 + 19 = 77

let primeSum4 = TotalSumOfPrimeLessThan 225286
// 2 + 3 + 5 + 7 + ........... = 2,147,431,330

let primeSumError = TotalSumOfPrimeLessThan 225287
// ERROR: integer overflow!
```

Notice that we cannot sum too many (prime) numbers, because the maximum range of `int` is $2^{31} - 1 = 2,147,483,647$. We will revisit this question later.

3.3 List.map

```
let Square x = x * x
let result9 = List.map Square [1 .. 10]
```

The `List.map` function transform each individual element of a list using some transformation. The transformation is specified through a function `Square`.



Alternatively, we can use the `fun` keyword to define the `Square` function

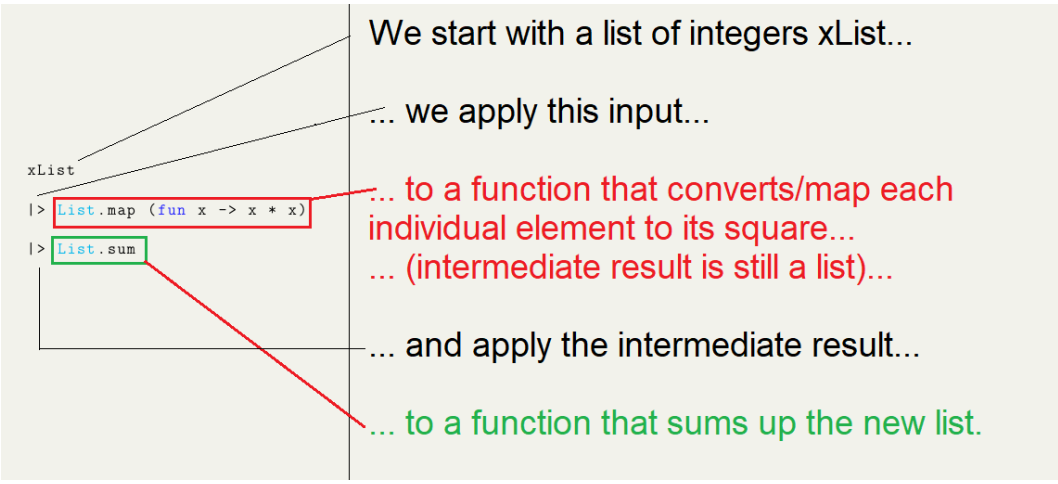
```
let result10 = List.map (fun x -> x * x) [1 .. 100]
```

List.map and Pipe-Forward |>

Let us look at an example:

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

// 1^2 + 2^2 + 3^2 + 4^2 + ... + 10^2 = 385
let result11 = SumOfSquares [1 .. 10]
```

How to interpret the code:



Exercise

There are two supermarkets in town. One of them want to round the prices of each individual goods to the nearest dollar (might round-up or round-down). The other want to round DOWN the prices of each individual goods to the nearest dollar.

The functions `System.Math.Floor, System.Math.Round*` are used round the prices:

```
let originalPrice1 = 1.35
let originalPrice2 = 3.99

let newPrice1 = originalPrice1 |> System.Math.Floor
let newPrice2 = originalPrice2 |> System.Math.Floor

// Temporary ignore decimal numbers like 1.50, 2.50.
let newPrice3 = originalPrice1 |> System.Math.Round
let newPrice4 = originalPrice2 |> System.Math.Round
```

Output:

```
val newPrice1 : float = 1.0
val newPrice2 : float = 3.0

val newPrice3 : float = 1.0
val newPrice4 : float = 4.0
```

*Remark: We will temporary ignore decimals like $1.50, 2.50$, because F# uses "Banker's Rounding" when tie-breaking is required. (Google it for more info)

1. Write a function that accepts a list of prices of the original products, and computes the final price of everything after each item are individually rounded-down.

```
// Round the prices to closest integer (ignore 1.50, 2.50,
    etc.)
let TotalPriceAfterRoundDown priceList =



    // Implement your function here.
```

2. Write a function that accepts a list of prices of the original products, and computes the final price of everything after each item are individually rounded to the nearest integer (ignore $1.50, 2.50$, etc.).

```
let TotalPriceAfterRound priceList =



    // Implement your function here.
```

Application: Sample Variance

We will try to implement the sample variance function (`VAR.S` in Excel 2010 or later, or see `https://www.miniwebtool.com/sample-variance-calculator/`).

$$\text{Sample Variance} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2$$

Remark: It is divided by $N-1$, not N, because of statistics reasons (Bessel's correction).

```
let SampleVariance xList =
    let N =
        xList
        |> List.length
        |> double          // The "double" function

    let average =
        xList
        |> List.average

    // return this:
    xList
    |> List.map (fun x -> (x - average) ** 2.0)
    |> List.sum
    |> fun final -> final / (N - 1.0)
```

Remark:

1. We use the `double` function to convert an integer to decimal (you can also use it to process `string` to decimals, if the `string` is well defined)
2. At the last step, we divide by `(N - 1.0)` and not `(N - 1)` because we are working with decimals.
3. The compiler knows xList is a `float list` or `List<float>`, because at some point it interacted with `** 2.0`.

```
let result12 = SampleVariance [1.0 .. 7.0]
// val result12 : double = 4.666666667
```

Exercise (Euler Project Question 6)

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

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

If you want, you can use the following hint:

```
let ProjectEulerProblem6 xList =
    // if xList = [a;b;c], calculate a^2 + b^2 + c^2
    let sumOfSquares =



    // if xList = [a;b;c], calculate a + b + c
    let sum =



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

To use the function:

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

Expeected answer: 25164150

Remark: After you have completed this question, you can create an account and submit your solution here for personal achievement/accomplishment.

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

Example (Euler Project Question 10 Revisited)

`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 $(2,000,000)$.

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

Hopefully, your solution before is the following (using the `IsPrime` function in the previous section):

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

However, we will encounter integer overflow when we add too many numbers, and exceed the range of `int` of $2^{31}-1 = 2,147,483,647$

```
let primeSum4 = TotalSumOfPrimeLessThan 225286
// 2 + 3 + 5 + 7 + ..... = 2,147,431,330

let primeSumError = TotalSumOfPrimeLessThan 225287
// ERROR: integer overflow!
```

Instead, after extracting out the prime numbers, we convert each prime number to `BigInteger` that can handle large sums using `List.map`

```
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
```

Remark: You can create an account and submit your solution here for personal achievement/accomplishment.
`https://projecteuler.net/problem=10`