This lab is designed to get you started with using F# within Visual Studio, and to solve a moderately complicated calculation problem by composing a number of simpler functions.

---

**Getting Started with F# in Visual Studio**

Reboot the machine to MS Windows if it is running under Linux, and log on.

Open up Visual Studio from the windows start menu.

In VS: File → New → Project → Visual F# → F# tutorial – then at the bottom change the name from Tutorial 1 to Lab1Calendar and click OK. This will show you some simple code demonstrating the features of F#. (You can use F# application instead in future labs.)

Select a line or two of the code and press Alt-Enter – the code will be sent to the F# interactive window, and the result displayed.

You can also type declarations and expressions directly into the F# interactive window, but you will need to add ;; at the end to indicate the end of each input.

Remove the tutorial code and rename Tutorial.fs as Lab1Calendar.fs. Then build the functions below, one by one, putting them in the Lab1Calendar.fs file. As you go, test each function by selecting the code for it, pressing Alt-Enter, and then running the function for some inputs. (You can include these tests in your file to avoid retyping them.)

**Lab details**

This lab involves calculating with dates: in particular, it involves determining the era-day of a given date, and conversely the date of a given era-day. The era-day of a date \(d\) is the number of days between 1 January 1 and \(d\). For example, the era-day of 1 January 1 is 1, the era-day of 31 December 1000 is 365242, and the era-day of 12 March 1999 is 729825.

1. A year \(y\) is a leap-year iff \(y\) is divisible by 400 or if it is divisible by 4 and not divisible by 100.

   Define a function \(\text{isleap} : \text{int} \rightarrow \text{bool}\) that takes a year \(y\) and tells us if \(y\) is a leap-year. For example, 1992 and 2000 are both leap-years; 1900 and 1901 are both not leap-years.

**Translating dates to era-days**

2. The era-day of the last day of year \(y\) is \(365 \times y + 1\) for each leap-year between 1 and \(y\). We can use the principle of inclusion and exclusion to count these leap-years. Add all the years that are multiples of 4; but this includes years like 1900 that aren’t leap-years, so subtract all the years that are multiples of 100; but this excludes years like 1600 that are leap-years, so add all the years that are multiples of 400.

   Define a function \(\text{daysToEndYear} : \text{int} \rightarrow \text{int}\) that takes a year \(y\) and returns the era-day of 31/12/y. E.g., \(\text{daysToEndYear} 1 = 365\) and \(\text{daysToEndYear} 1792 = 654515\).
3. The number of days between 1 January \( y \) and the end of month \( m \) in year \( y \) is given by \((367 \times m + 5)/12 - c\), where \( c \) is a correction to allow for the possibility that \( y \) may be a leap-year. The correction is 0 if \( m \) is January, 1 if \( m \) is after January and \( y \) is a leap-year, and 2 if \( m \) is after January and \( y \) is not a leap-year. 

Define a function \( \text{daysToEndMonth} : \text{int*int} \rightarrow \text{int} \) that takes a month \( m \) and a year \( y \) and returns the era-day of last/m/y. E.g.,

\[
\text{daysToEndMonth}(9,1792) = 654423 \\
\text{daysToEndMonth}(7,622) = 227027.
\]

4. Define a function \( \text{eraDay} : \text{int*int*int} \rightarrow \text{int} \) that takes a date and returns its era-day. E.g.,

\[
\text{eraDay}(1,1,1) = 1, \\
\text{eraDay}(19,7,622) = 227015, \\
\text{eraDay}(22,9,1792) = 654415 \text{ and } \text{eraDay}(12,3,1999) = 729825.
\]

Translating era-days to dates

5. Given an era-day \( x \), calculating the year in which \( x \) falls is quite involved. We actually want the year up until the end of the previous day (e.g., day 365 is still in year 1), so we start with \( x-1 \).

a) Each 400-year period has the same number of days (400*365+97), so first count the number of complete 400-year periods in \( x-1 \).

b) \( y \) days are left over: count the number of complete 100-year periods (100*365+24 days) in \( y \).

[Special case: if this yields 4 periods then there's actually only 3.]

c) \( z \) days are left over: count the number of complete 4-year periods (4*365+1 days each) in \( z \).

d) \( w \) days are left over: finally, count the number of complete years (365 days each) in \( w \).

[Special case: if this yields 4 periods then there's actually only 3.]

The result is the sum of the years in all these periods, plus 1 (because the first year is 1, not 0).

Define \( \text{yearOf} : \text{int} \rightarrow \text{int} \) that takes an era-day \( x \) and returns the year in which \( x \) falls.

E.g.,

\[
\text{yearOf} 1 = 1, \quad \text{yearOf} 1461 = 4, \quad \text{yearOf} 1462 = 5, \\
\text{yearOf} 227015 = 622, \quad \text{yearOf} 654415 = 1792, \quad \text{yearOf} 729825 = 1999.
\]

6. Given an era-day \( x \), the month in which \( x \) falls is given by \((12(z+c) + 373)/367\), where \( y \) is the year in which \( x \) falls, \( z \) is the number of days between 1/1/y and \( x \), and \( c \) is a correction to allow for the possibility that \( y \) may be a leap-year. If \( y \) is a leap-year, the correction is 0 if \( z<60 \) and 1 if \( z\geq60 \). If \( y \) is not a leap-year, the correction is 0 if \( z<59 \) and 2 if \( z\geq59 \).

Define \( \text{monthOf} : \text{int} \rightarrow \text{int} \) that takes an era-day \( x \) and returns the month in which \( x \) falls.

E.g.,

\[
\text{monthOf} 1 = 1, \quad \text{monthOf} 1461 = 12, \quad \text{monthOf} 1462 = 1, \\
\text{monthOf} 227015 = 7, \quad \text{monthOf} 654415 = 9, \quad \text{and } \text{monthOf} 729825 = 3.
\]

7. Define \( \text{dateOf} : \text{int} \rightarrow \text{int*int*int} \) that takes an era-day \( x \) and returns the date of \( x \). E.g.,

\[
\text{dateOf} 1 = (1,1,1), \quad \text{dateOf} 1462 = (1,1,5), \quad \text{dateOf} 227015 = (19,7,622), \quad \text{dateOf} 654415 = (22,9,1792), \quad \text{dateOf} 729825 = (12,3,1999).
\]

An application

8. Define a function \( \text{daysAfter} : \text{int} \rightarrow \text{int*int*int} \rightarrow \text{int*int*int} \) that takes a number of days \( x \) and a date \( d \) and returns the date \( x \) days after \( d \). For example,

\[
\text{daysAfter} 366 (12,3,1999) = (12,3,2000).
\]

Note: we haven't fully covered multi-argument functions like this, but for now all you need to know is that you can define them as you'd expect:

\[
\text{let daysAfter n (d,m,y) } = \quad // \text{Put your code here}
\]