4.4 The Calendar program

To illustrate the power of functions, in this section we will develop a useful program that allows the user to input a date or a month or a year. For a specific date the program will respond with the day of the week on which this date occurred or will occur. For example, 2 16 1952 was a Saturday (and my birthday) while 12 7 1941 was a Sunday (and the day on which Pearl Harbor was attacked). If the user inputs a month the program will respond with a calendar for that month. For a year the program responds with 12 calendars for the 12 months of that year. This is a substantial program with 14 different functions, but its development is rather straightforward.

Our program is based on one fact and a simple idea. The fact is that January 1, 1800 was a Wednesday. We use this as our starting date because it comes after the adoption of the Gregorian calendar by England and its colonies in 1752. Wednesday, September 2 1752 was followed by Thursday, September 14 1752 to bring the civil calendar more in line with the seasons. There have been no significant changes to the Western calendar since.

The idea we will use is to notice that if the number of days between two dates is divisible by 7 then these dates must occur on the same day of the week. For example, August 12 and August 26 differ by 14 days; in 2009 they both occur on a Wednesday while in 2005 they both are on a Friday. For our purposes "days between" will mean all days including the first date but not including the second, so the 7 days between August 12 and August 19 are numbered 12, 13, 14, 15, 16, 17, and 18. Our algorithm comes down to counting the number of days between January 1, 1800 and the date we are interested in, and taking this count mod 7. If the result is 0 the date occurs on a Wednesday, if it is 1 the date occurs on a Thursday, and so forth.

The first step is to get input from the user. We know the form of an input loop; we'll read dates and respond to them until we get a signal that the user is done with the program. But how do we read a date? There are lots of possible formats. We could ask the user for a year, then a month in that year, then a date in that month, but this seems very awkward and slow. A more intuitive format from the user's point of view would be to enter all of the information on one line: a user could input

```
9 16 2009
```

to learn that September 16, 2009 occurs on a Wednesday. We will store the input values in a list, where the individual entries are numbers are numbers. Lists are written inside parentheses, as in [9, 16, 2009]. The algorithm for reading the date is similar to a lot of Python input algorithms. We will first read the entire line of input, because reading a line of text as a string is easy. There is a string method called `split()` that divides a string into "words", using white space as the delimiter between words. If `s` is the line of input, `s.split()` is a list of the words (which in our case are the numbers) of `s`. A **for**-loop can run through this list; we can use the **int**() function to convert each word into its numeric value. Altogether, we get the following function for reading a date:

```
def ReadDate():
    s = input( "Enter a date in the form m d y, \
or a blank to exit: " )
    t = []
    for x in s.split():
        t = t + [int(x)]
    return t
```

This function always returns variable t, which starts as an empty list, and has numbers added to it. If there is a no input, as will happen when the user enters a blank line, function readDate() returns an empty list. We can use this as the exit condition for our program.

Our main() function is our usual input loop, with format:

```
def main():
    done = False
    while not done:
        date = ReadDate()
        if  len(date) == 0:
           done = True
        else:
            <print the day for this date>
```

We will assign a function PrintDay() to the task of printing the day of the week for a particular date. The information that needs to be sent to this function are the month, day, and year of the date: these are the first, second, and third fields that function ReadDate() returns:

```
def main():
    done = False
    while not done:
        date = ReadDate()
        if  len(date) == 0:
            done = True
        elif len(date) == 3:
            PrintDay(date[0], date[1], date[2])
        else:
            print( "Bad date." )
```

Function PrintDay() holds the basic activity of this program: it takes as arguments a month, day, and year, and prints the day of the week corresponding to this date. We will divide this into two steps. The hard part is computing the numeric day of the week: 0 for Sunday, 1 for Monday, and so forth. We'll create a function FindDay() for this. Once we have that number, we will give it to a simple function DayName() to print as a string. This makes function PrintDay() particularly easy:

```
def PrintDay(m, d, y):
    # This prints the day of the week (m,d,y) falls on
    day = FindDay(m, d, y)
    print( "That was a %s" % DayName(day) )
```

The DayName() function is also easy:

```
def DayName(d):
    #This returns a string for the name of day d
    if d == 0:
        return "Sunday"
    elif d == 1:
        return "Monday"
    elif d == 2
        ...
        <and so forth>
```

The real work is in writing function FindDay(). This starts by finding the number of days between January 1, 1800 and day (m, d, y); we will us a function CountDays() to compute this. We will store this count in variable days. We will include January 1, 1800 in this count but not day (m, d, y). Recall that January, 1, 1800 fell on a Wednesday; every 7 days after this we are back to a Wednesday. This means that if days%7 is 0, then (m, d, y) also falls on a Wednesday. On the other hand, if days % 7 is 1, then (m, d, y) falls on a Thursday; if it is 2 then (m, d, y) is a Friday, and so forth. Here is the code that comes from this analysis.

```
def FindDay(m, d, y):
    # This counts the number of days between (1,1,1800)
    # and (m,d,y), takes the remainder %7, and
    # returns 0 for Sundays, 1 for Mondays, etc.
    days = CountDays(m, d, y)
    day = days%7
    if day == 0:
        return 3  # Wednesday is day 3 of the week
    elif day == 1:
        return 4 # Thursday
        ...
    elif day == 3:
         return 6   # Saturday
    elif day == 4:
        return 0   # Sunday
       ...
<and so forth>
```

Function CountDays() is easier than you might think. Remember that this function counts the number of days between January 1, 1800 and day (m, d, y). We can break this into three steps:

- The sum of all the days of all the years starting with 1800, up to but not including year y (we don't want to include all of year y). We keep a running total of these, adding 365 for non-leap years and 366 for leap years.

- The sum of all the days of all the months of year y starting with January, up to but not including month m. We add each of these onto the total.

- The number of days of month m up to, but not including day d.

The first two of these we accomplish with a loop; the third by adding d−1 onto our total. The following code does this; we use helper-functions to compute the number of days in a year (365 if it isn't a leap year, 366 if it is) and the number of days in a month (31 for January, 28 for non-leap year Februarys, 29 for leap year Februarys, 31 for March, etc.):

```
def CountDays(m, d, y):
    # This returns the number of days between
    # (1, 1, 1800) and (m,d,y).
    days = 0
    for year in range(1800, y):
        days = days + DaysInYear( year )
    for month in range(1, m):
        days = days + DaysInMonth(month, y)
    days = days + d-1
    return days
```

Notice how we can use the ability of a computer to execute many small steps quickly to solve a problem in a way that human would think of solving it. Part of the art of programming is learning to make use of this ability.

The rest of the program is completely straightforward. The complete program is given below. This looks like a lot of code, but the development of it was not difficult. At each step we write the code for one small portion of the program. If this is more than just a few lines of code, we break it into pieces and assign a function to write each piece. Note that we write the call of the function, and then go back and write the function itself. As long as we have a clear vision of what each function should do (comments help us to keep that straight), the individual pieces will all fit together this way we designed them precisely to do that.

```
# This program finds the day of the week corresponding to
# any specific (m,d,y) date after January 1, 1800.

def IsLeapYear(y):
    # This returns True if y is a leap year, False otherwise
    if y % 400 == 0:
        return True
    elif y % 100 == 0:
        return False
    elif y % 4 == 0:
        return True
    else:
        return False

def DaysInMonth(m, y):
    # This returns the number of days in month m of year y.
    # Of course, the year only matters for February
    if m == 1:
        return 31
    elif m == 2:
        if isLeapYear(y):
            return 29
        else:
            return 28
    elif m == 3:
        return 31
    elif m == 4:
        return 30
    elif m == 5:
        return 31
    elif m == 6:
        return 30
    elif m == 7:
        return 31
    elif m == 8:
        return 31
    elif m == 9:
        return 30
    elif m == 10:
        return 31
    elif m == 11:
        return 30
    elif m == 12:
        return 31
    else:
        print( "daysInMonth: No such month %d" % m )
```

Program 4.4.1: The Calendar Program - beginning

```
def DayName(d):
    #This returns a string for the name of day d
    if d == 0:
        return "Sunday"
    elif d == 1:
        return "Monday"
    elif d == 2:
        return "Tuesday"
    elif d == 3:
        return "Wednesday"
    elif d == 4:
        return "Thursday"
    elif d == 5:
        return "Friday"
    elif d == 6:
        return "Saturday"

def DaysInYear(year):
    # This returns the number of days in the given year.
    if IsLeapYear(year):
        return 366
    else:
        return 365

def CountDays(m, d, y):
    # This returns the number of days between
    # (1, 1, 1800) and (m,d,y)
    days = 0
    for year in range(1800, y):
        days = days + DaysInYear( year )
    for month in range(1, m):
        days = days + DaysInMonth(month, y)
    days = days + d-1
    return days
```

Program 4.4.1: The Calendar Program continued

```
def FindDay(m, d, y):
    # This counts the number of days between (1,1,1800)
    # and (m,d,y), takes the remainder %7, and returns
    # 0 for Sundays, 1 for Mondays, etc.
    days = CountDays(m, d, y)
    day = days%7
    if day == 0:
        return 3
    elif day == 1:
        return 4
    elif day == 2:
        return 5
    elif day == 3:
        return 6
    elif day == 4:
        return 0
    elif day == 5:
        return 1
    elif day == 6:
        return 2
    else:
        print( "FindDay: bad day %d" % day )

def PrintDay(m, d, y):
    # This prints the day of the week (m,d,y) falls on
    day = FindDay(m, d, y)
    print( "That was a %s" % DayName(day) )

def ReadDate():
    # This asks the user for a date, then makes a tuple
    # of numbers out of the user's response.
    print( "Either enter a date in the form m d y" )
    s = input( "or enter a blank to exit: " )
    t = []
    for x in s.split():
        t = t + [int(x)]
    return t

def main():
    done = False
    while not done:
        date = ReadDate()
        if  len(date) == 0:
            done = True
        elif len(date) == 3:
            PrintDay(date[0], date[1], date[2])
        else:
            print( "Bad date." )

main()
```

Program 4.4.1: The Calendar Program continued