# COMP 1012 Winter 2015 Assignment 3

## *Due Date: Tuesday, February 24, 2015, 11:59 PM*

New Material Covered

- `for` loops
- `lists` and `tuples`, with subscripts and slices
- Unicode strings
- formatting

Notes:

- Name your script file as follows: `<LastName><FirstName>A3Q1.py.` For example, `LiJaneA3Q1.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA3Q1V2.py` is a valid name for Rob Smith's file.
- Follow the posted programming standards to avoid losing marks.
- You must submit a ***Blanket Honesty Declaration*** to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully. You will upload both script file and output via a dropbox on the course website. We will demonstrate the assignment hand-in procedure in lectures. There will be a period of several days before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment within 48 hours ***after*** the deadline, it will be accepted, but you will receive a penalty (roughly 1% per hour).

## *Question 1—Thirty Days Hath September [14 marks]*

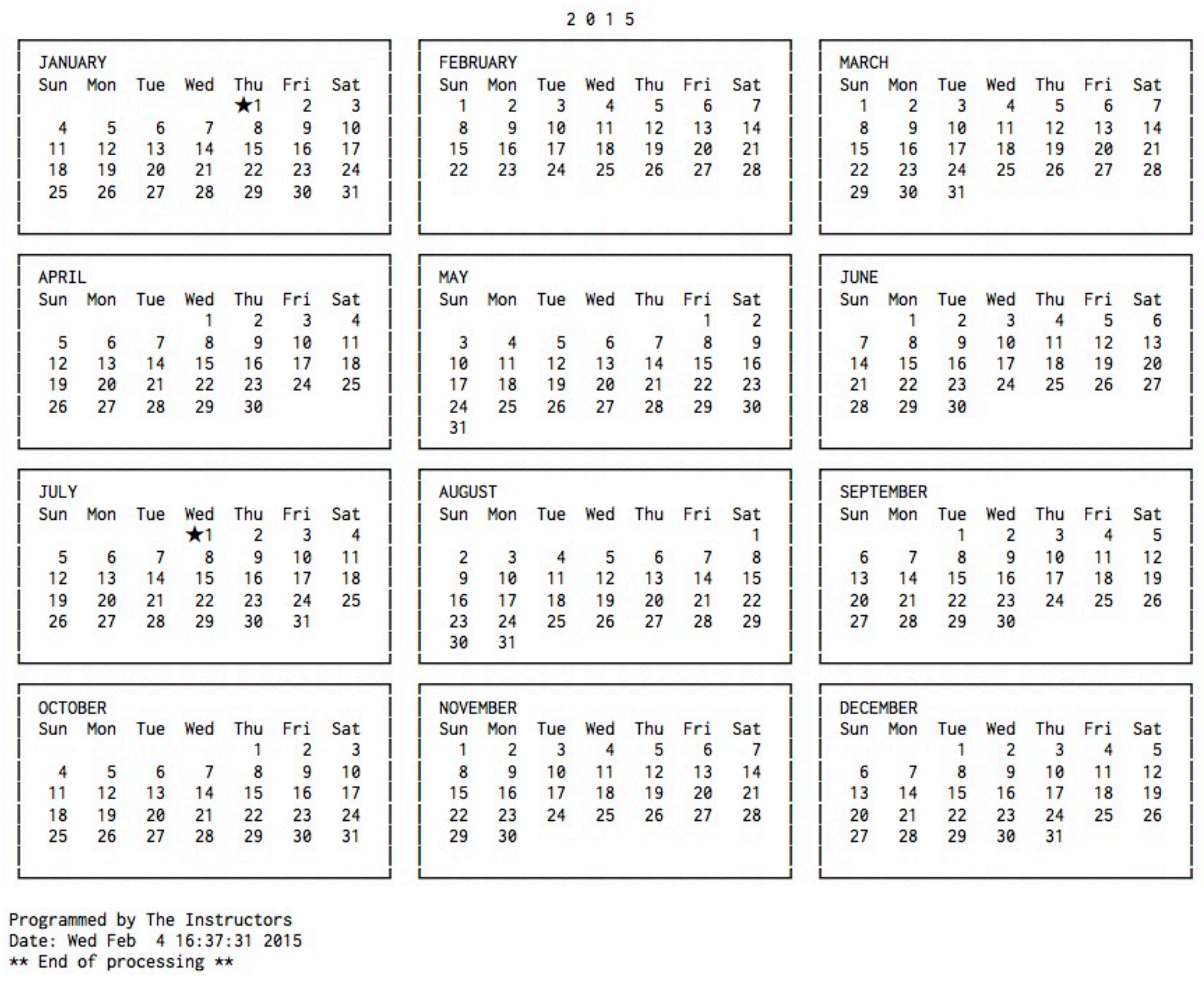
### *Description*

Write a script that can print a full year's calendar for any year from 1900 to 9999, using Excel dates, as described back in Assignment 1. For example, successful output for 2015 would look as shown on the next page. A calendar should have the following feature:

- The year, double-spaced, centered across the top.
- 12 months, arranged in a 4×3 array, each with a month name, a row of day abbreviations, and 6 rows of dates, with the appropriate days of the month shown lined up in the correct columns.
- Each month should be surrounded by a dark border.
- Two days should be highlighted with a large black star (not an asterisk), namely New Year's Day (January 1) and Canada Day (July 1).
- Termination output should be shown after the calendar.

### *What to do*

Write a script that prints the calendar. It should contain several function definitions, specified below. The instructions following the function definitions should look something like the boxed instructions on the next page. By changing the value assigned to `YEAR` in this script you can generate a calendar for any year you specify. Alternatively you may prompt the user for the value of year and use `raw_input`. The loop generates and prints a monthly calendar for each month in turn. The call to `theEnd` prints the termination message identifying the programmer and giving the date.



```
                                        2 0 1 5

 JANUARY                             FEBRUARY                            MARCH
 Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat
                     ★1    2    3     1    2    3    4    5    6    7     1    2    3    4    5    6    7
  4    5    6    7    8    9   10     8    9   10   11   12   13   14     8    9   10   11   12   13   14
 11   12   13   14   15   16   17    15   16   17   18   19   20   21    15   16   17   18   19   20   21
 18   19   20   21   22   23   24    22   23   24   25   26   27   28    22   23   24   25   26   27   28
 25   26   27   28   29   30   31                                        29   30   31

 APRIL                               MAY                                 JUNE
 Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat
                  1    2    3    4                            1    2          1    2    3    4    5    6
  5    6    7    8    9   10   11     3    4    5    6    7    8    9     7    8    9   10   11   12   13
 12   13   14   15   16   17   18    10   11   12   13   14   15   16    14   15   16   17   18   19   20
 19   20   21   22   23   24   25    17   18   19   20   21   22   23    21   22   23   24   25   26   27
 26   27   28   29   30              24   25   26   27   28   29   30    28   29   30
                                     31

 JULY                                AUGUST                              SEPTEMBER
 Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat
                 ★1    2    3    4                                 1               1    2    3    4    5
  5    6    7    8    9   10   11     2    3    4    5    6    7    8     6    7    8    9   10   11   12
 12   13   14   15   16   17   18     9   10   11   12   13   14   15    13   14   15   16   17   18   19
 19   20   21   22   23   24   25    16   17   18   19   20   21   22    20   21   22   23   24   25   26
 26   27   28   29   30   31         23   24   25   26   27   28   29    27   28   29   30
                                     30   31

 OCTOBER                             NOVEMBER                            DECEMBER
 Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat   Sun  Mon  Tue  Wed  Thu  Fri  Sat
                      1    2    3     1    2    3    4    5    6    7               1    2    3    4    5
  4    5    6    7    8    9   10     8    9   10   11   12   13   14     6    7    8    9   10   11   12
 11   12   13   14   15   16   17    15   16   17   18   19   20   21    13   14   15   16   17   18   19
 18   19   20   21   22   23   24    22   23   24   25   26   27   28    20   21   22   23   24   25   26
 25   26   27   28   29   30   31    29   30                             27   28   29   30   31

Programmed by The Instructors
Date: Wed Feb  4 16:37:31 2015
** End of processing **
```

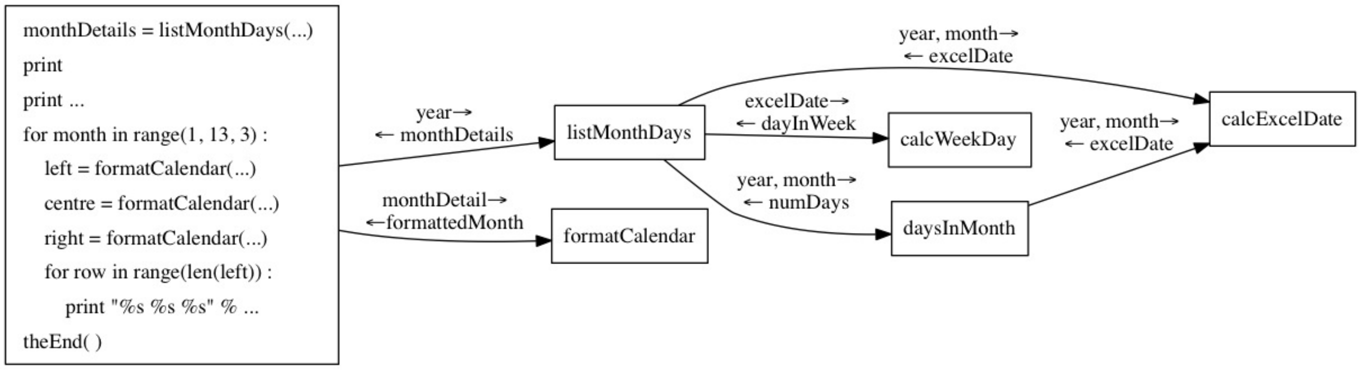
## *Instructions to Draw the Calendar*

```
YEAR = 2015                         # year to make the calendar
monthDetails = listMonthDays(YEAR)  # details for each month
print …                             # details of year print statement omitted

for month in range(1, 13, 3) :
    left =   formatCalendar(monthDetails[month]).split('\n')
    centre = formatCalendar(monthDetails[month+1]).split('\n')
    right =  formatCalendar(monthDetails[month+2]).split('\n')
    for row in range(len(left)) :
        print "%s %s %s" % (left[row], centre[row], right[row])

theEnd( )
```

### *Functions*

The structure of your functions should look as shown on the next page in the call graph. The main body of your script is shown on the left side. It calls `listMonthDays` and passes the `year`, getting back `monthDetails`, described below. The script also calls `formatCalendar` once for each month, to set up a monthly calendar, which is returned as `formattedMonth`. The function `listMonthDays` in turn calls `calcExcelDate`, `calcWeekDay` and `daysInMonth`. A call graph like this can be a handy tool in figuring out the big picture—how the program fits together.

## *Call Diagram*



Details of function definitions, in alphabetical order:

- `calcExcelDate(year, month)`: This function implements the same formula used in Assignment 1 to calculate an Excel date. The year and month only are provided; assume the day is 1 and that there are no fractions of days. You may copy your code from Assignment 1, or you may copy the code from the sample solution to Assignment 1. In either case, adapt it to run in a function. The Excel date is returned as an integer from this function call.
- `calcWeekDay(excelDate)`: Given an integer Excel date, this function returns an int from 0 to 6 indicating which day of the week the Excel date refers to, where 0 designates Sunday, and 6 designates Saturday. If you just take the Excel date mod 7, you will get a number from 0 to 6, but it won't be quite right. Try a few days and look for a slightly more complicated pattern that will produce the right result.
- `daysInMonth(year, month)`: The number of days in the given month is the difference between the Excel date of the first day of this month and the Excel date of the first day of the next month. Return this integer. Do not just store the numbers of days in each month, since the number does differ from year to year, at least for February.
- `formatCalendar(monthDetail)`: The parameter `monthDetail` is the tuple generated by `listMonthDays` for a single month. The month name is in the first position of the tuple, and the second position contains a list of 42 3-character fields to appear in the calendar. Return a single character string that provides the multi-line calendar needed for a single month. Put a newline character at the end of each line. The dark border comes from the following Unicode characters, which are two characters wide in some fonts:.[1]
    - `U"\N{BOX DRAWINGS HEAVY DOWN AND RIGHT}"`
    - `U"\N{BOX DRAWINGS HEAVY UP AND RIGHT}"`
    - `U"\N{BOX DRAWINGS HEAVY HORIZONTAL}"`
    - `U"\N{BOX DRAWINGS HEAVY DOWN AND LEFT}"`

[1] You may encounter some difficulty getting characters to line up correctly. Typically Canopy uses a fixed width font for both the editor and Python pane. In such a font, every normal character is exactly the same width. However, Unicode characters beyond the basic ASCII characters have a wide variety of widths. It is recommended that you set the font to *Inconsolata* (Preferences ... General ... Font), because the line drawing characters and the star character have widths that are easy to line up. When you hand in your output, take a screenshot of your screen instead of copying from the Python pane, as the marker may have trouble getting the borders of the monthly calendars to line up the same way they do for you.

  - `U"\N{BOX DRAWINGS HEAVY UP AND LEFT}"`
  - `U"\N{BOX DRAWINGS HEAVY VERTICAL}"`
- `listMonthDays(year)`: This function returns `monthDetails`, a list of 13 items.
  - The first item, in position 0, is the year that was passed down. The remaining 12 items are the details of the months 1 to 12 for this particular year. For each month, the details are stored in a tuple with two entries. The first entry in the tuple is the full name of the month. The second entry in the tuple is a list of 3-character strings showing what is to be printed in the 42 positions of the monthly calendar (6 weeks × 7 days per week). For example, the details for the month of January in 2015 would be this tuple:
    `('JANUARY', ['   ', '   ', '   ', '   ', '★1', '  2', '  3', '  4',`
    `… ' 30', ' 31', '   ', '   ', '   ', '   ', '   ', '   ', '   '])`
  - The month names should come from a constant tuple.
  - You can calculate the number of blank entries to start each list by calling `calcWeekDay` with the `excelDate` of the first day of the month, obtained by calling `calcExcelDate`. The function `calcWeekDay` returns a number from 0 to 6 representing Sunday through Saturday. If you know the first day of the month is on a Monday (day 1), for example, then clearly there will be 1 blank day before it. Enough blank days are added at the end to ensure the total length of the list of days is 42.
  - In January and July, replace `'  1'` by `'★1'`. The star character can be obtained by using a Unicode string `U"\N{BLACK STAR}"`. Notice that this single character replaces two blanks, because it is twice as wide as a blank in many fonts.
- `theEnd()`: This function has no parameters. It prints the three lines of output to show the program is terminating shown in the sample output.

## Hand-in

Hand in the script file used to produce the calendar. Follow correct naming conventions and follow the Programming Standards rules. Also hand in the output for two cases: 2015 and 2016. In each case, take a screenshot showing the output, save it as a jpeg or png file, and hand in both these files.