10.009 The Digital World

Term 3. 2016

Problem Set 5 (for Week 5)

Last update: December 24, 2015

- **Problems: Cohort sessions**: Following week: Monday 11:59pm.
- **Problems: Homework**: Same as for the cohort session problems.
- **Problems: Exercises**: These are practice problems and will not be graded. You are encouraged to solve these to enhance your programming skills. Being able to solve these problems will likely help you prepare for the midterm examination.

Objectives

1. Learn modularity.
2. Learn how to divide complex problems into smaller modules.
3. Learn recursion.
4. Learn how to create custom modules.

Note: Solve the programming problems listed below using the IDLE or Canopy editor. Make sure you save your programs in files with suitably chosen names and in an newly created directory. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems: Cohort sessions

1. *Game: Craps:* Craps is a popular dice game played in casinos. Write a program to play a variation of the game, as follows: Roll two dice. Each die has six faces representing values 1, 2, . . ., and 6, respectively. Check the sum of the two dice. If the sum is 2, 3 or 12 (called craps), you lose; if the sum is 7 or 11 (called natural), you win; if the sum is another value (i.e. 4,5,6,8,9, or 10), a point is established. Continue to roll the dice until either a 7 or the same point value is rolled. If 7 is rolled, you lose. Otherwise, you win. Your program acts as a single player. Here are some sample runs:

```
You rolled 5 + 6 = 11
You win

You rolled 1 + 2 = 3
You lose

You rolled 4 + 4 = 8
point is 8
You rolled 6 + 2 = 8
You win

You rolled 3 + 2 = 5
point is 5
You rolled 2 + 5 = 7
You lose
```

2. *Calendar year:* The goal of top-down design is that each module provides clearly defined functionality, which collectively provide all of the required functionality of the program. The three overall steps of the calendar year program are getting the requested year from the user, creating the calendar year structure, and displaying the year. Modules "display calendar year" are not too complex and need not be broken down further. The module "construct calendar year", however, is where most of the work is done. We need to break down this module into several functionalities:

   - `def leapYear(year)`: Returns True if provided year is a leap year, otherwise returns False. Check `http://en.wikipedia.org/wiki/Leap_year#Algorithm`.
   - `def dayOfWeekJan1(year)`: Returns the day of the week for January 1 of the provided year. `year` must be between 1800 and 2099. The return value must be in the range 0-6 (0-Sun, 1-Mon, . . ., 6-Sat). Check `http://en.wikipedia.org/wiki/`

`Determination_of_the_day_of_the_week#Gauss.27_algorithm`. The weekday of the first of January in year number A is given by:

$$d = R(1 + 5R(A-1, 4) + 4R(A-1, 100) + 6R(A-1, 400), 7)$$

where $R(y, x)$ is a function that returns the remainder when y is divided by x. In Python, it is similar to executing `y % x`.

- `def numDaysInMonth(month_num, leap_year)`: Returns the number of days in a given month. `month_num` must be in the range 1-12, inclusive. `leap_year` must be True if month in a leap year, otherwise False.
- `def constructCalMonth(month_num, first_day_of_month, num_days_in_month)`: Returns a formatted calendar month for display on the screen. `month_num` must be in the range 1-12, inclusive. `first_day_of_month` must be in the range 0-6 (0-Sun, 1-Mon, ..., 6-Sat). Returns a list of strings of the form,

  ```
  [month_name, week1, week2, ...,]
  ```

  For example, the first two weeks of January 2015 will be

  ```
  ['January,'             1  2  3,' 4  5  6  7  8  9 10']
  ```

  As the first two weeks for January 2015 will be displayed as

  ```
             1  2  3
   4  5  6  7  8  9 10
  ```

  If the number of days of the last week is less than seven, no spaces are added after the last date. For example, the last week of December 2015 will be

  ```
  '27 28 29 30 31'
  ```

  Notice that the number of days is five days and there are no spaces added after the characters 31.

- `def constructCalYear(year)`: Returns a formatted calendar year for display on the screen. `year` must be in the range 1800-2099, inclusive. Returns a list of strings of the form,

  ```
  [year, month1, month2, month3, ..., month12]
  ```

  in which each month sublist is of the form

  ```
  [month_name, week1, week2, ...,]
  ```

The other main function is:

- `def displayCalendar(calendar_year)`: Displays the provided `calendar_year` on the screen. Provided calendar year should be in the form of a list of twelve sublists, in which each sublist is of the form

  `[month_name, week1, week2, ...,]`

You should test the individual functions separately. Once each functions are properly working, do the integration testing. Calling `displayCalendar` will display the calendar from January to December for that particular year. An example for two of the months are shown below.

```
March
 S  M  T  W  T  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31


April
 S  M  T  W  T  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

To test, run the following Python script:

```
print 'Start'
ans=displayCalendar(2015)
print ans
print 'End'
```

The output should be:

```
Start
January
 S  M  T  W  T  F  S
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

February
```

```
 S  M  T  W  T  F  S
 1  2  3  4  5  6  7
 .
 .
 .

December
 S  M  T  W  T  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
End
```

3. *Recursion:* Write a function that takes in a number n and returns its factorial. You can solve this problem either using loops or recursion. Note that $0! = 1$, and $n! = n \times (n-1)!$ for $n > 0$.

Problems: Homework

1. *Modular Design:* Implement a set of functions called `getData`, `extractValues`, and `calcRatios`. Function `getData` should prompt the user to enter pairs of integers, two per line, with each pair read as a single string. For example,

   ```
   Enter integer pair (hit Enter to quit):
   134 289 (read as '134 289')
   etc
   ```

   These strings should be passed one at a time as they are being read to function `extractValue`, which is designed to return the string as a tuple of two integer values,

   ```
   extractValues('134 289') returns (134, 289)
   etc
   ```

   Finally, each of these tuples is passed to function `calcRatios` one at a time to calculate and return the ratio of the two values. For example,

   ```
   calcRatios((134,289)) returns 0.46366782006920415
   ```

   When the second value of the tuple is zero, the ratio is not defined, and the function should return `None`.

   Tutor Submission: Submit only the function definitions for extractValue and calcRatios.

2. *Modular: Calendar* Add another functionality to the Calendar Year Program to prompt the user to choose either to display a complete calendar year or just a specific calendar month. If user choose to display a specific calendar month, the program should call another function that prompts the user to enter which month it is. Modify `displayCalendar` to take in another parameter which is the specific month. If None is specified on this parameter, it means that the function should print the complete calendar.

Problems: Exercises

1. *Recursion:* Write a function called `moveDisks(n, fromTower, toTower, auxTower,sol)` to solve the Towers of Hanoi (Check `http://en.wikipedia.org/wiki/Tower_of_Hanoi`) problem recursively. `n` is the number of disks, the next three parameters are labels of the towers, and the last one is a list to contain the solutions. The function should return a list of strings. For example,

```
moveDisks(1,'A','B','C',sol)
sol=['Move disk 1 from A to B']

moveDisks(3,'A','B','C',sol)
['Move disk 1 from A to B', 'Move disk 2 from A to C',
'Move disk 1 from B to C', 'Move disk 3 from A to B',
'Move disk 1 from C to A', 'Move disk 2 from C to B',
'Move disk 1 from A to B']
```

End of Problem Set 5.