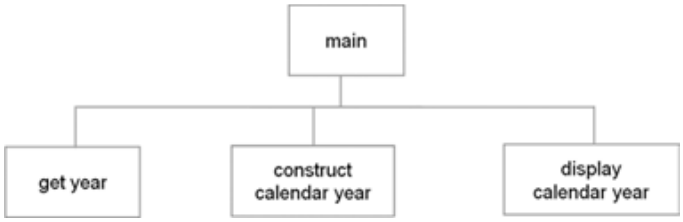
Top-Down Design

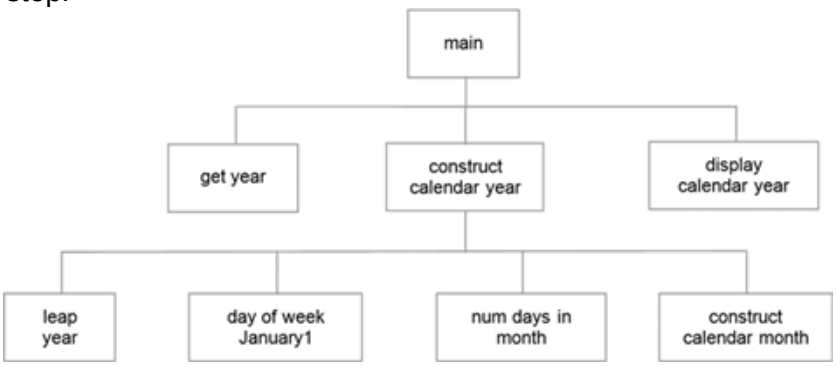
One method of deriving a modular design is called **top-down design**. In this approach, the overall design of a system is developed first.

Developing a Modular Design of the Calendar Year Program

You will now develop a modular design for the calendar year program from Unit 4 (implemented there without the use of functions) using a top-down design approach. The three overall steps of the program are getting the requested year from the user, creating the calendar year structure, and displaying the year. This is shown here:



Then consider whether any of these modules needs to be further broken down. Making such a decision is more of an art than a science. The goal of modular design is that each module provides clearly defined functionality, which collectively provide all of the required functionality of the program. Modules get year and display calendar year are not complex enough to require further breakdown. Module construct calendar year, on the other hand, is where most of the work is done, and is therefore further broken down. The figure below contains the modules of this next design step.



In order to construct a calendar year, it must be determined whether the year is a leap year, what day of the week January 1st of that year falls on, and how many days are in each month (accounting for leap years). Thus, modules leap year, day of week January 1, and num days in month are added as *submodules* of module construct calendar year. The calendar month for each of the twelve months must then be individually constructed, handled by module construct calendar month.

Calendar Year Program Modules

The modular design of the calendar year program provides a high-level view of the program. However, there are many issues yet to resolve in the design. Since each module is to be implemented as a function, we need to specify the details of each function. For example, for each function it needs to be decided if it is a value-returning function or a non-value-returning function; what parameters it will take; and what results it will produce. We give such a specification on the next page using Python docstrings.

```
def getYear():
    """ Returns an integer value between 1800-2099, inclusive, or  -1."""

def leapYear(year):
    """ Returns True if provided year a leap year, otherwise returns False."""

def dayOfWeekJan1(year, leap_year):
    """ Returns the day of the week for January 1 of the provided year.

        year must be between 1800 and 2099. leap_year must be True if
        year a leap year, and False otherwise.
    """

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 (1-Sun, 2-Mon, ..., 0-Sat).

        Returns a list of strings of the form,
        [month_name, week1, week2, week3, week4, week5, week6].
    """

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 list of strings of the form,

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

def displayCalendar(calendar_year):
    """ Displays the provided calendar_year on the screen three months across.

        Provided calendar year should be in the form of a list of twelve sublists, in which each
        sublist is of the form [monthname, week1, week2, ..., ]
    """
```

Notes:

This stage of the design provides sufficient detail from which to implement the program. The main module provides the overall construction of the program. It simply displays the program greeting, calls module `getYear` to get the year from the user, calls module `constructCalYear` to construct the year, and finally calls module `displayCalendar` to display the calendar year. The only detail that the main module is concerned with is allowing the user to keep displaying another calendar year, or enter -1 to terminate the program. This is controlled by Boolean variable terminate.

The first module called, `getYear`, returns the integer value entered by the user. The module's specification indicates that it returns an integer value between 1800 and 2099, inclusive; or 21 (if the user decides to terminate the program). Therefore, it is the responsibility of the module to ensure that no other value is returned. This relieves the main module of having to check for bad input.

The next module that the main module uses is module `constructCalYear`. This module returns a list of twelve sublists, one for each month, in which each sublist begins with the name of the month (as a string value), followed by each of the weeks in the month, each week formatted as a single string. The module's specification indicates that it is to be passed a year between 1800 and 2099, inclusive. Therefore, any year given to it that is outside that range violates its condition for use, and therefore the results are not guaranteed.

The last module called from the main module is module `displayCalendar`. It is given a constructed calendar year, as constructed by module `constructCalMonth`. Based on the modular design of the calendar year program, `constructCalYear` is the only module relying on the use of submodules, specifically modules leapYear, `dayOfWeekJan1`, `numDaysInMonth`, and `constructCal`. The leapYear module determines whether a given a year is a leap year or not, returning a Boolean result. Module `dayOfWeekJan1` returns the day of the week for January 1st of the provided year. Boolean value leap_year must also be provided to the module, needed in the day of the week algorithm on which the module is based. Module `numDaysInMonth` must be passed an integer in the range 1–12, as well as a Boolean value for leap_year. This is so that the module can determine the number of days in the month for the month of February. Finally, `constructCalMonth` is given a month number, the day of the week of the first day of the month (1-Sun, 2-Mon, . . ., 0-Sat) and the number of days in the month. With this information, module `constructCalYear` can construct the calendar list and its sublists to be displayed.

Finally, module `displayCalendarMonth` is given a formatted calendar year. Its job is to display the calendar year three months across.

Concepts and Procedures

1. In top-down design (select one),

a) The details of a program design are addressed before the overall design.

b) The overall design of a program is addressed before the details.

2. All modular designs are a result of a top-down design process. (TRUE/FALSE)

3. In top-down design, every module is broken down into the same number of submodules. (TRUE/FALSE)

4. Which of the following advantages of modular design apply to the design of the calendar program.

a) Provides a means for the development of well-designed programs.
b) Provides a natural means of dividing up programming tasks.
c) Provides a means of separately testing individual parts of a program.

Programming Problems

1. 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
etc.
```

Implement a complete program that displays a list of ratios for an entered series of integer value pairs. Make sure to include docstring specification for each of the functions.