1 Stars

1.a Reverse Stars

This question requires nested loops. Part(a) is a straightforward modification of what we have seen in the lab. Remember, the number of times the for loop body is repeated is equal to the number of elements in the range. For the jth row, any range which contains \( r - j + 1 \) elements should work. I picked \( j:r \).

```matlab
r = input('Enter the number of rows: '); for j = 1:r
    for k = j:r
        fprintf('*');
    end
    fprintf('
'); end
```

1.b Diamond

The most intuitive solution would be to combine what we have done for the stars in the lab and in this homework. For the top part of the diamond, you can use the same for loop to print spaces just like reverse stars, but followed by stars similar to the exercise in the lab. For the bottom part of the diamond spaces become like the lab exercise and stars are like a modification of the problem above. The number of elements in the ranges determine how many stars or space will be printed. For example the following code should print diamonds with odd widths:

```matlab
w = input('Enter the width of the diamond: '); r = (w-1)/2;
for j = 1:(r+1)
    for k = j:r
        fprintf(' '); end
    for k = 1:(2*j-1)
        fprintf('*'); end
    fprintf('
'); end
for j = 1:r
    for k = 1:j
        fprintf(' '); end
    for k = 1:(w-2*j)
        fprintf('*'); end
    fprintf('
'); end
```
fprintf('*');
end
fprintf('n');
end

In order to check whether the width is odd or not, and keep asking for input if it is even, you can use a while loop. This is similar to the revisited version of the number game script when we learned while loop.

Besides, the diamond has a symmetric shape, we can make use of negative numbers in our for loop ranges, and the absolute value function to print top and bottom parts at once. Here is a better version:

```
while true
    w = input('Enter diamond width: ');
    if rem(w+1,2)
        disp('Please enter an odd number!');
    else
        break;
    end
end
r = (w-1)/2;
for j = -r:r
    for k = 1:abs(j)
        fprintf(' ');
    end
    for k = 1:(w-2*abs(j))
        fprintf('*');
    end
    fprintf('n');
end
```

## 2 Calendar

### 2.a Generator

We will write a function called `generate_calendar`. It has one input, the first day of January as a string, and one output, a matrix with entries 0-7. I will call the output `cal` and the input `jan1`.

```
function cal = generate_calendar(jan1)
```

Next task is to determine which number we will start with. The user input will be a string, and we have to set a variable to the corresponding number provided in the homework pdf. It can be done neatly with `switch/case`. We will use variable `j` to keep track of 0-7.
switch jan1
  case 'Monday' , j = 1;
  case 'Tuesday' , j = 2;
  case 'Wednesday', j = 3;
  case 'Thursday' , j = 4;
  case 'Friday' , j = 5;
  case 'Saturday' , j = 6;
  case 'Sunday' , j = 7;
end

Another way would be to use if/elseif/end statements and strcmp function which was mentioned in Hw2 Solutions.

if strcmp(jan1,'Monday')
  j = 1;
elseif strcmp(jan1,'Tuesday')
  j = 2;
elseif strcmp(jan1,'Wednesday')
  j = 3;
elseif strcmp(jan1,'Thursday')
  j = 4;
elseif strcmp(jan1,'Friday')
  j = 5;
elseif strcmp(jan1,'Saturday')
  j = 6;
elseif strcmp(jan1,'Sunday')
  j = 7;
end

which is equivalent to writing

if strcmp(jan1,'Monday'), j = 1;
elseif strcmp(jan1,'Tuesday'), j = 2;
elseif strcmp(jan1,'Wednesday'), j = 3;
elseif strcmp(jan1,'Thursday'), j = 4;
elseif strcmp(jan1,'Friday'), j = 5;
elseif strcmp(jan1,'Saturday'), j = 6;
elseif strcmp(jan1,'Sunday'), j = 7;
end

We can initialize the output cal to zeros. We only need to care about the days which already exist, the rest will remain 0 if we don’t touch them.

cal = zeros(12,31);

Now we know which number we will start and have a matrix filled with zeros. We will use a nested loop to fill in the entries. The outer loop will visit the rows which represent the months, and the inner loop will visit the columns which represent the days of the month. However, the range of the inner loop is dependent on the row, just like the Stars. This time it doesn’t follow a simple rule, but we know their limits. We can make use of an array with size 12 for
each month.

```matlab
days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];
```

And lastly we fill in the values. We have to keep track of the variable \( j \) and reset it back to 1, after 7.

```matlab
for m = 1:12
    for d = 1:days_in_month(m)
        cal(m,d) = j;
        j = j + 1;
        if j == 8, j = 1; end
    end
end
```

If you don’t use a list of values like `days_in_month`, and have fixed ranges `1:12`, `1:31`, then you have to use `if/end` to check invalid dates. We don’t want to increment the variable \( j \) or update the matrix entry when we visit an invalid date.

```matlab
if is_valid_date
    do stuff
end
```

Expressing the invalid date is easier

```matlab
is_invalid_date = ((m==2 && (d==29 || d==30)) || ... 
                 (m==2 || m==4 || m==6 || m==9 || m==11) && d==31);
```

```matlab
if ~is_invalid_date
    do stuff
end
```

Notice the `NOT` operator `~`. We don’t use the variable `is_invalid_date` anywhere else, so I just will use it directly as the condition.

```matlab
for m = 1:12
    for d = 1:31
        if ~(m==2 && (d==29 || d==30)) || ... 
            (m==2 || m==4 || m==6 || m==9 || m==11) && d==31)
            cal(m,d) = j;
            j = j + 1;
            if j == 8, j = 1; end
        end
    end
end
```

Here is a full listing of the function using switch/case and days in the month:
function cal = generate_calendar(jan1)

switch jan1
    case 'Monday' , j = 1;
    case 'Tuesday' , j = 2;
    case 'Wednesday', j = 3;
    case 'Thursday' , j = 4;
    case 'Friday' , j = 5;
    case 'Saturday' , j = 6;
    case 'Sunday' , j = 7;
    otherwise
        error('Invalid name for a day!');
end

cal = zeros(12,31);
days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];

for m = 1:12
    for d = 1:days_in_month(m)
        cal(m,d) = j;
        j = j + 1;
        if j == 8, j = 1; end
    end
end

2.b Error Function

The otherwise code fragment in switch/case, and else section of the if/else/end can be used to capture erroneous input. We can display an error message but we also want to stop running the script. error function can be used to accomplish this task. The simplest usage can be just like the disp function, which takes a string input.

switch jan1
    case 'Monday' , j = 1;
    case 'Tuesday' , j = 2;
    case 'Wednesday', j = 3;
    case 'Thursday' , j = 4;
    case 'Friday' , j = 5;
    case 'Saturday' , j = 6;
    case 'Sunday' , j = 7;
    otherwise
        error([jan1 ' is not a day name!']);
end

How can we repeat the erroneous input in our message? Remember that a string is an array of characters, and we can expand arrays.

otherwise
    error(['Invalid name for a day!']);
end
2.c  Which Day

We need to convert the month, represented by a string into a number to use as a row index. The \texttt{day} is already provided as an integer, so we can directly use it as a column index. After we loop up our calendar for a value in $0 - 7$, we can generate a string to display with another \texttt{switch/case} statement.

\begin{verbatim}
function d = which_day(cal, month, day)
    switch month
        case 'January', m = 1;
        case 'February', m = 2;
        case 'March', m = 3;
        case 'April', m = 4;
        case 'May', m = 5;
        case 'June', m = 6;
        case 'July', m = 7;
        case 'August', m = 8;
        case 'September', m = 9;
        case 'October', m = 10;
        case 'November', m = 11;
        case 'December', m = 12;
    end
    d = cal(m,day);
    switch d
        case 0, name = ' doesn''t exist!';
        case 1, name = ' is Monday.';
        case 2, name = ' is Tuesday.';
        case 3, name = ' is Wednesday.';
        case 4, name = ' is Thursday.';
        case 5, name = ' is Friday.';
        case 6, name = ' is Saturday.';
        case 7, name = ' is Sunday.';
    end
    fprintf('%s %d %s
', month, day, name);
\end{verbatim}

2.d  Monthly Calendar

Monthly calendar requires looping over the days of the month, and keeping track of the weekdays to print a newline. First we have to convert the input string, \texttt{month}, into a row index, as we did in \texttt{which_day}. The main difficulty might be to print the first week of the month, since we have to know how many days to skip (print space characters). You can solve this problem in various ways. You can write for loops to print spaces, or based on the start day you can have a \texttt{switch/case} and use \texttt{disp} or \texttt{fprintf} to display the first week. The code provided below uses \texttt{for} loops to skip the first few days if required.
function monthly_calendar(cal, month)

switch month
    case 'January', m = 1;
    case 'February', m = 2;
    case 'March', m = 3;
    case 'April', m = 4;
    case 'May', m = 5;
    case 'June', m = 6;
    case 'July', m = 7;
    case 'August', m = 8;
    case 'September', m = 9;
    case 'October', m = 10;
    case 'November', m = 11;
    case 'December', m = 12;
    otherwise
        error(['month, ' is not a month name!']);
end

fprintf(' %s
', month);
fprintf('Su Mo Tu We Th Fr Sa
');

% How many days should we skip on the first row?
first = cal(m,1);
skip = rem(first,7);

% Let's skip them
for j = 1:skip
    fprintf('    '); 
end

% Let's complete the first row
for day = 1:(7-skip) % as long as the day exists
    fprintf('%2d ', day);
    day = day + 1;
    j = j + 1;
    if j == 8
        fprintf('
');
        j = 1;
    end
end

fprintf('
');
2.e Leap Years

The common knowledge is that February has 29 days if the year is divisible by 4. The less common knowledge is that there are exceptions to this rule which you will learn by experience if you can live long enough. At the turn of the century, it doesn’t apply! You might say in year 2000, there were 29 days in February. Well, there is another exception too. If the year is divisible by 400, February will have 29 days, like in 2000. If you’d checked the blog article by Moler on *February the 13th*, you might have noticed that he has defined a function to determine if a year is leap year or not. We can add a similar variable `isleap` in our script, and set the second value in the array `days_in_month` to 29. The following code implements the additional input `year` and creates the calendar matrix appropriately.

```matlab
function cal = generate_calendar(jan1, year)
    cal = zeros(12,31);
    switch jan1
        case 'Monday' , j = 1;
        case 'Tuesday' , j = 2;
        case 'Wednesday', j = 3;
        case 'Thursday' , j = 4;
        case 'Friday' , j = 5;
        case 'Saturday' , j = 6;
        case 'Sunday' , j = 7;
        otherwise
            error(['jan1', ' is not a day name!']);
    end

    days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];

    % Check if it is a leap year
    if nargin > 1
        if mod(year,400) == 0, isleap = true;
        elseif mod(year,100) == 0, isleap = false;
        elseif mod(year, 4) == 0, isleap = true;
        else isleap = false; end

        if isleap, days_in_month(2) = 29; end
    end

    for m = 1:12
        for d = 1:days_in_month(m)
            cal(m,d) = j;
            j = j + 1;
            if j == 8, j = 1; end
        end
    end
```
3 More Taxicab

3.a Two More Loops

Following the solution strategy in Hw2 Solutions, we can assume $a$ to be the smaller of $(a, b)$, and add two more loops in between $p$ and $q$ as follows. But when you run this script, you won’t be able to get a result unless you are extremely(!) patient. Try to avoid too many nested loops at all cost! There is no point in going more detail of such a slow process.

```matlab
function minS = taxicab3 (n)
if nargin < 1
    n = 500;
end
minS = 2*n^3;
for x = 1:n
    for p = x+1:n
        for a = p+1:n
            for b = a+1:n
                S1 = a^3 + b^3;
                for q = b+1:n
                    S2 = p^3 + q^3;
                    if S1 == S2
                        for y = q+1:n
                            if S2 == x^3 + y^3
                                if minS > S2, minS = S2; end
                            end
                        end
                    end
                end
            end
        end
    end
if minS < 2*n^3
    printf('Minimum Taxicab(3) is %d\n', minS);
end
```

3.b Fast Taxicab

The lecture slides from July 11 are appended by a script which used a matrix to create a look up table. The code generates the look up table using a nested for loop. And since the table is symmetric, takes only the upper half and converts it into a one dimensional array. Then if finds the unique elements in this array. If there are any repetitions it means there is a taxicab in our range. The last part finds how many repetitions exist, if ask for the ones which are repeated twice we get Taxicab(2). If you change 2 to 3 you will get the Taxicab(3)
numbers. The smallest Taxicab(3) is 87539319.

```matlab
function S = fast_taxicab3 (n)
    if nargin < 1
        n = 450;
    end
    S = zeros(n);
    for x = 1:n;
        for y = x:n; % there is no need to generate the lower part
            S(x,y) = x^3 + y^3;
        end
    end
    values = [];
    % this could also be done in an efficient way
    % without expanding an array. think about it.
    for x = 1:n  % without expanding an array. think about it.
        values = [values, S(x,x+1:end)];
    end
    results = [];
    for j = unique(values)  % all you had to do was to change 2 --> 3
        if length(find(values == j)) == 3
            results = [results, j];
        end
    end
    S = min(results);
end
```