1.1 Not string but chars

In MATLAB, there is the type char but not the type string. What we call a string is really an array of chars. Type each of the following statements in the Command Window and note the result.

```
a = pi;     % A numeric scalar
b = 'pi';  % A char array. Use SINGLE quotes to enclose a char or multiple chars
```

```
c = length(b)  % __________ b is an array, so one can use function length on it
d = ['apple' 'b' 'es']  % Vector concatenation. d should be the string 'apple pies'
e = [d; 'muffin']  % _________________________________
```

```
e = [d; 'muffins ' ]  % Note the two extra 'm's and one trailing space
[nr,nc]= size(e)  % __________ e is a matrix, so one can use function size on it
f= e(1, 7:9)  % _________________________________ Accessing a subarray
e(1, 7:10)= 'core'  % _________________________________
```

```
g= ones(2,3)*67;  % A NUMERIC 2-by-3 matrix, each component has the value 67
h= char(g)  % _________________________________
i= double(h)  % _________________________________
```

```
jj= char(floor(rand*26) + 'A')  % _________________ A random upper case letter
k= jj>'a' && jj<'z'  % __________ True or false: character stored in jj is lower case
L= strcmp('abcd', 'ab')  % _____________________________ strcmp compares the arguments
m= 'abcd'=='ab'  % ERROR: attempted vectorized code on vectors of different lengths
n= 'abcd'=='abCd'  % ________________________________ Vectorized code—result is a vector
```

```
o= sum('abcd'=='abCd')  % __________________________ The number of matches
n= sum('abcd'=='abCd')  % __________________________ The number of mismatches
```

1.2 Reverse complement

In the DNA double helix, two strands twist together and "face" each other. The two strands are reverse-complementary, i.e., reading one strand in reverse order and exchanging each base with its complement gives the other strand. A and T are complementary; C and G are complementary.

For example, given the DNA sequence `AGTACGCAT`  the reverse sequence is `TACGATGA` so the reverse complement is `ATGCTACT`

Write a function `rCompBulk(mat)` to return the reverse complements of a set of DNA strands. `mat` is a matrix of characters; each row of the matrix represents one strand of DNA (so `mat` contains only the letters 'A', 'T', 'C', and 'G'). Return a matrix the same size as `mat` such that the `r`th row of the returned matrix is the reverse complement of the `r`th strand of DNA (the `r`th row of `mat`). Use loops—do not use vectorized code.
2 Subarrays

Type the following expressions in the MATLAB Command Window. Write the resulting array or answer the question on the blank.

\[
\begin{align*}
m &= \text{rand}(6,5) \\
a &= m(:,2) & \% \text{What does the colon specify when used in place of an index?} \\
b &= m(2:3,:) & \% \text{____________________________} \\
p &= \text{rand}(6,5,3) & \% \text{This is a 3-dimensional array} \\
\text{[nr, nc, np]} &= \text{size}(p) & \% \text{____________________________} \\
c &= p(:,:,2) & \% \text{Is this a matrix (2-d) or a 3-d array?} \\
d &= p(4,:,2) & \% \text{Is this a vector, matrix, or 3-d array?} \\
\end{align*}
\]

3 Two-dimensional interpolation

When you enlarge an image, you are actually adding data points among the existing data (pixels). How do you get the additional data points? One way is to interpolate from the neighboring points—take the average value. First, consider a simple case of one-dimensional interpolation, we add a data point between neighboring pairs of existing data points by taking the simple average. For example,

\[
\begin{bmatrix}
2.0 & 1.0 & 1.0 & 2.0 \\
\end{bmatrix}
\]

becomes

\[
\begin{bmatrix}
2.0 & 1.5 & 1.0 & 1.0 & 1.0 & 1.5 & 2.0 \\
\end{bmatrix}
\]

In 2-d interpolation, work with one dimension at a time. For example, given a matrix

\[
\begin{bmatrix}
2.0 & 1.0 & 1.0 & 2.0 \\
6.0 & 5.0 & 4.0 & 3.0 \\
5.0 & 5.0 & 5.0 & 4.0 \\
\end{bmatrix}
\]

First we can add a column between two neighboring columns, so the matrix becomes 3 × 7:

\[
\begin{bmatrix}
2.0 & 1.5 & 1.0 & 1.0 & 1.0 & 1.5 & 2.0 \\
6.0 & 5.5 & 5.0 & 4.5 & 4.0 & 3.5 & 3.0 \\
5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 4.5 & 4.0 \\
\end{bmatrix}
\]

Then add a row between neighboring rows, so the final matrix will be 5 × 7:

\[
\begin{bmatrix}
2.0 & 1.5 & 1.0 & 1.0 & 1.5 & 2.0 \\
4.0 & 3.5 & 3.0 & 2.8 & 2.5 & 2.5 & 2.5 \\
6.0 & 5.5 & 5.0 & 4.5 & 4.0 & 3.5 & 3.0 \\
5.5 & 5.2 & 5.0 & 4.8 & 4.5 & 4.0 & 3.5 \\
5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 4.5 & 4.0 \\
\end{bmatrix}
\]

Write two versions of the following function Interpolate2D: (a) use non-vectorized code; (b) use vectorized code (work with whole rows, or whole columns, one at a time). Do not use built-in function \text{linspace}.

\[
\begin{align*}
\text{function newM = Interpolate2D(M)} \\
&\% \text{Perform 2-d interpolation on the real-valued data in nr-by-nc matrix M.} \\
&\% \text{The interpolated data are added between existing data points so newM is} \\
&\% \text{(2*nr-1)-by-(2*nc-1). Use the simple average as the interpolated value.}
\end{align*}
\]

Please delete your files from the computer before leaving the lab.