Function Name: `luge`

Inputs:

1. (*char*) A hyphen-separated list of lugers and their times

Outputs:

1. *(char*) A sentence stating the faster luger and their time

Banned Functions:

`find(), strfind(), textscan()`

Background:

Working behind the scenes at the Olympics can be super stressful—billions of people watch and wait in suspense to see who wins each event. You're in charge of finding the fastest time for the luge race. The pressure is on to make sure you find the true winner. Don't mess up! Thankfully, you can use your MATLAB skills to ensure you don't cause another *La La Land* Oscars fiasco...

Function Description:

Given an input character vector that contains a list of lugers and their times, iterate to find the luger with the fastest time. All inputs will be in the following format:

```
'<name1>-<time1>-<name2>-<time2>-...-<nameZ>-<timeZ>'
```

Each name is paired with a time (in seconds). All entries are separated by hyphens. Once you find the luger with the fastest time, output a character vector in the following format:

```
'<fastestName> is the winner of the 2018 Olympics for luge, with a time of
                              <time> seconds!'
```

Example:

```
>> str = luge('Albert Demchenko-208.00-Felix Loch-207.531-Armin
          Zoeggeler-208.80')
str = 'Felix Loch is the winner of the 2018 Olympics for luge, with a
       time of 207.53 seconds!'
```

Notes:

- The input can have any number of names and times, but will always have at least one name/time pair.
- Names and times will always be separated by a single hyphen.
- No two times will be the same.
- Round the time to the second decimal place only when formatting the output string.

Hints:

- The `strtok()` function will be useful.
- Use `%0.2f` with the `sprintf()` function to round the time to the second decimal place.

Function Name: `olympicTorch`

Inputs:

1. (*char*) MxN character array representing a map
2. (*char*) 1xP character vector representing a path to take

Outputs:

1. *(logical*) Whether the given instructions lead you to your destination

Background:

It's 1984: the world is changing, and so are you. You decide to better yourself and train to be an Olympic torch relay runner. The night before the big day, however, you stay up too late working with this hot new software tool and you oversleep your alarm. Scrambling out your door, disoriented and half-awake, you can't remember who you are or where you're supposed to be. Fortunately, you manage to grab your map, but the directions are torn off! A bystander manages to decipher your nonsensical gargling and gives you a set of directions, but you can't be sure whether to trust them; after all, you're not even sure you heard them correctly...

Function Description:

Write a function called `olympicTorch()` that will determine whether a given path brings you to the destination on the given map.

Example:

Here is an example of a map that might be used as the first input:

```
#######
#..  *#
#.. ..#
#   ..#
# ....#
#0....#
#######
```

The map will be surrounded by a border, represented with hash characters (`'#'`). The single starting point (which can be anywhere on the map) is represented by the `zero` character (`'0'`), and the single destination is represented as an asterisk character (`'*'`). Areas that are not part of the path are filled in with period characters (`'.'`).

Continued...

Here is an example of a path:

```
'uduurruurlrr'
```

The characters in the path string represent the direction in which you should move:

`'u'` - go one space up,
`'r'` - go one space right,
`'d'` - go one space down,
`'l'` - go one space left.

Start at the `'0'` character. If the example path is followed, the destination is reached at the end of the path. Since no period or hash characters were crossed, the example path is a valid path.

If the path is valid, output a logical `true`, and if the path is not valid, output a logical `false`.

Notes:

- A path will be valid if and only if it does not cross over any period or hash characters and ends up on the destination (the asterisk).
- A path could, at some point, cross over the ending point but ultimately end up somewhere else (in which case it would **not** be valid).
- The directions may lead outside of the map.

Hints:

- You can use the second output of `find()` to determine the starting location.

Function Name: `zamboni`

Inputs:

1. (*char*) The file name of the input image, including file extension

Outputs:

None

File Outputs:

1. The output image with `'blur_'` appended to the beginning of the filename

Banned Functions:

All image functions except `imread()` and `imwrite()` are banned for this problem.

Background:

The International Olympics Committee needs you to smooth out the ice before the big curling event! They need you to drive the zamboni over all of the ice in the rink. In order to show the Committee that you did your job, you will show what the design under the ice looks like, but more blurry than before.

Function Description:

Write a function called `zamboni()` that takes in an input image and outputs the same image modified with a *box blur*. In a box blur operation, each center pixel becomes the average of all 9 pixels around it (including itself), for every color layer. Refer to the example operation below for a 3x3 `uint8` array.

```
30 30 30          ** ** **
90 90 90    ->    ** 60 **
60 60 60          ** ** **
```

To determine the color values of the pixels in the output image, follow these steps:

1) Index out a 3x3 piece of the original array, where the middle pixel of the 3x3 piece corresponds to the new pixel.
2) Average the values in the 3x3 array, then set the new pixel to the average value (Do this for each layer of the image separately).
3) Repeat steps 2) and 3) for all 3x3 pieces of the image array.

There are not enough pixels in the original image to calculate the border values. Hence, the output image will have 2 fewer rows (top and bottom rows) and 2 fewer columns (far left and right columns) than the original. Write the resultant image with `'blur_'` appended to the beginning of the input image filename.

Continued...

Example:

```
>> zamboni('spongebobImg.png')
```

spongebobImg.png

blur_spongebobImg.png

Notes:

- Check out this link for more information about kernels and image processing.
- You should **not** overwrite pixels in the original image as you iterate. Use the original pixel values to determine your new output pixels.
- The solution file will produce file outputs named `'blur_<original filename>_soln.png'`

Function Name: `pascalsPodium`

Inputs:

1. (*double*) The number of rows to include in your output

Outputs:

1. *(double)* An array representing the podium

Banned Functions:

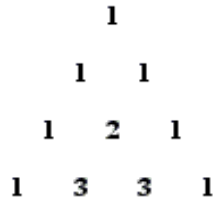
`nchoosek(), pascal()`

Background:

You have been hired by the International Olympic Committee to design the closing ceremonies. You and your team have all the flower bouquets picked out, ordered the medals, and purchased all the streamers and confetti, but still have one extremely important task: designing the podium. Reminiscing back to high school calculus, you remember learning about Pascal's triangle and decide it would be the perfect starting point to constructing the Olympic podium. However, mapping out rows and rows of blocks to recreate Pascal's triangle can be tedious, so you choose to use your amazing MATLAB skills to do it for you.

Function Description:

Given the number of rows of Pascal's triangle to include in your final output, build a double array containing the triangle, with zeros filling the spaces in between. For example, the first four rows of Pascal's triangle are usually written in the following fashion:



Since the rows of the triangle are offset from one another, our MATLAB representation of the triangle will include zeros to fill the gaps. So we can show the first four rows of the triangle as the following 4x7 double array:

0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	2	0	1	0
1	0	3	0	3	0	1

Each row of Pascal's triangle is constructed by summing adjacent elements of the row above it. For example, to find the value of a certain position, go to the position directly above it (the previous row) and sum the values directly to the right and left of it.

Continued…

Given this, you can create the array by starting with the first row (it will always be a single 1 in the center of the row), and iterating through the rest of the rows, using the elements of the previous row to create each subsequent row.

Notes:

- Inputs to this function will be integers greater than 1.

Function Name: `roundRobin`

Inputs:

1. (*char*) An Nx15 character array of game scores

Outputs:

1. *(char*) A string describing the outcome of the round robin

Background:

The National Hockey League (NHL) made news this Olympic season by announcing that it will not be allowing its players to participate in the PyeongChang games. This move has set the stage for an exciting men's tournament where a country besides Canada stands a chance of winning. In anticipation for the finals, you decide to revisit the excitement of the round robin games.

Function Description:

Round robin is a tournament format where each team plays a game against every other team in the bracket. Given a character array of the games results, write a function called `roundRobin()` that outputs a character vector describing the winner of the round robin. Each row in the character array represents one game, formatted in the following way:

```
'00 ABC - DEF 00'
```

`'ABC'` and `'DEF'` are the teams with their three-letter IOC country codes. Before each three-letter code are the goals scored by the team (always displayed as two digits). The team that wins the most games is the winner of the round robin. The output should be formatted

```
'Team <ABC> won the round robin with a record of <W>-<L>.'
```

where `<ABC>` is replaced by the IOC country code, `<W>` are wins, and `<L>` are losses.

Example:

```
>> arr = '02 USA - OAR 01
          12 USA - SVK 00'

>> result = roundRobin(arr)

result = 'Team USA won the round robin with a record of 2-0.'
```

Continued...

Notes:

- No game will end in a tie.
- There can be any number of teams or games played.
- There is guaranteed to be one team in first place.

Hints:

- Think about how you could use a character vector to keep track of the teams.

Extra Credit

Function Name: `cryptex`

Inputs:

1. (*char*) Filename of the cryptex image, including file extension
2. (*char)* Keyword

Outputs:

None

File Outputs:

1. Solved cryptex image (with `'_solved.png'` appended at the end of the filename)

Background:

Think you're as smart as Dr. Robert Langdon from the DaVinci Code? Let's put it to the test. If you succeed, you may even be able to enter the next Mind Sports Olympiad.

Function Description:

Write a function that takes in an image of a cryptex puzzle and solves it using the keyword provided. The cryptex should only be shifted column-wise (up/down) until the desired keyword is shown on the middle row of the cryptex.

Every letter in the cryptex will be 20 x 20 in image dimensions. Hence, if the following cryptex image is provided, then the image dimensions will be 100 x 80.

`test.png`

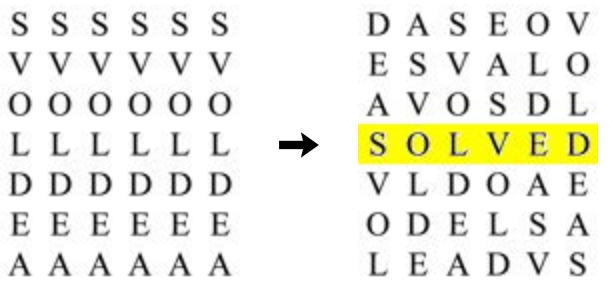
To help you with this problem, a helper function called `letterChecker()` has been provided. `letterChecker()` takes in a 20x20x3 `unit8` image array and outputs which letter it is, as a single character.

Using this tool, shift the columns of the cryptex image until the keyword is shown on the middle row of the image. Finally, to show that you have solved the cryptex, "highlight" the middle row by converting the white background to pure yellow. Output the solved cryptex image with `'_solved.png'` appended to the end of the filename.

Continued...

Example:

```
>> filename = 'puzzle.png'
>> key = 'SOLVED'
cryptex(filename,key)
```



`puzzle.png` `puzzle_solved.png`

Notes:

- The second input will never be lowercase.
- There will always be an odd number of letters in each column.
- There is guaranteed to be a single occurrence of a letter in each cryptex image column.
- `charStruct.mat` is a mat file used by `letterChecker()`. **Do not** `load` it.
- **Do not** use the `resize()` function. You do not need it.
- If you are having trouble visualizing this problem, refer to this video solving a replica cryptex from the DaVinci Code: https://youtu.be/11TLS3bOCt4?t=31
- The solution will output image files named '`<original filename>_solved_soln.png`'

Hints:

- Refer back to `puzzleBox()` from HW05.
- The `find()` and `circshift()` functions might be useful.