Drill Problem #1

Function Name: arrReplace

Inputs:

1. (*double*) An MxN array
2. (*double*) Another MxN array
3. (*double*) A number

Outputs:

1. (*double*) The original MxN array with replacements made

Function Description:

Write a function called arrReplace() that inputs two arrays and a number. The function will replace all instances of that number in the first array with the values in the corresponding positions of the second array. If the number does not exist in the array, it should just return the original array with no changes.

Example:

```
arr1 => [11,  8;      arr2 => [5,   2;          num = 8
          8,  4]               7,   6]


result = arrReplace(arr1, arr2, num)
result => [11, 2;
           7, 4]
```

Notes:

- The two input arrays are guaranteed to have the same dimensions.

Hints:

- When using logical operators with an array, it returns an array of logicals. This is called a mask. Think how this may be useful in solving this problem.

Drill Problem #2

Function Name: asciiResize

Inputs:

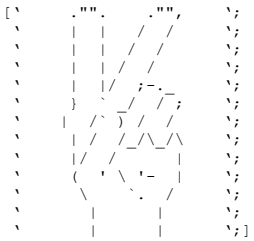
1. (*char*) An M x N array of characters
2. (*double*) A positive value greater than or equal to 1 specifying the scale factor

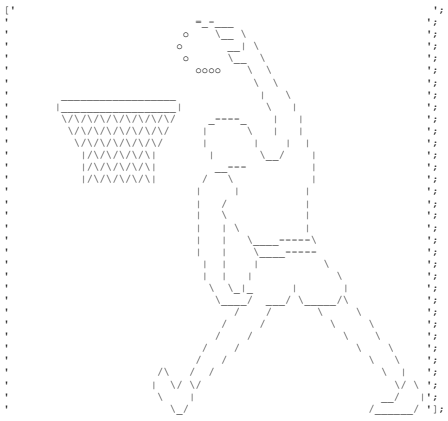
Outputs:

1. (*char*) The resized character array of the input

Function Description:

All nerds unite! ASCII art has evolved throughout the past few decades from a simple peace sign, like this: to Air Jordan, like this:





Write a MATLAB function called `asciiResize()` that takes in an MxN array of such ASCII art, and resizes it by the given scale factor. So if the input array was MxN and the scale factor was 3, the resulting array would be 3M x 3N.

Notes:

- The same scale factor is applied to the horizontal and vertical dimensions.
- If the scale factor is not an integer, you should round the third input of `linspace()` to obtain the number of new rows and columns.
- Included in the test cases are `'.mat'` files. Typing `load('<file_name>.mat')` into the Command Window, or double-clicking the `'.mat'` file in the Current Directory, will load any variables saved in the file to the Workspace.
- Included ASCII art creation courtesy of the internet.

Hints:

- Consider editing the `vecResize()` function (from Homework 04) and using it as a helper function to find the needed resized indices in a given dimension (rows versus columns).

Drill Problem #3

Function Name: howFall

Inputs:

1. (*double*) An MxN array of leaves in a tree
2. (*double*) A prediction of days left in Fall

Outputs:

1. (*logical*) A logical value indicating if the prediction was correct
2. (*char*) A string describing the leaves still hanging in the tree

Function Description:

The fall semester has begun, and soon the green leaves at Georgia Tech will turn yellow to orange to brown before finally falling off their tree. Your friend makes a prediction for the number of days left in the "Actual Fall" until the leaves have all dwindled away. Since you are a MATLAB connoisseur now, you decide to write a MATLAB program that will calculate if your friend's prediction is correct and will output a string with the number of green, yellow, orange and brown leaves still hanging after the prediction time.

Each position in the input array will correspond to a "leaf" in the tree, and the number will indicate how long until the leaf will fall off of the tree (this number is guaranteed to be an integer). The color of the leaf is proportional to this number and can be found according to the given table.

Color	Days Before the Leaf Falls Off
Brown	1-3
Orange	4-9
Yellow	10-14
Green	15 and above

After the predicted time has passed, if 20% of the leaves remain, the prediction was correct and the first output will be `true`. If more than 20% of the leaves are still hanging on the tree, the prediction was incorrect, and the first output should be `false`. Finally, the second output should be a string listing the number of green, yellow, orange, and brown leaves remaining, in the following format: `'There will be # leaves remaining: # green, # yellow, # orange, and # brown.'` Where the '`#`' is replaced with the actual number of each type of leaf remaining.

For example, given the following array and your friend predicting 5 days until fall:

```
tree = [10, 30; 12, 5]
```

The following array represents the new leaves and the days until they fall off, once the prediction has been applied:

```
newTree = [5, 25; 7, 0]
```

Since 75% of the tree's leaves are still hanging, your friend's prediction is `false`. Thus, the output string will be the following: `'There will be 3 leaves remaining: 1 green, 0 yellow, 2 orange, and 0 brown.'`

Notes:

- The 20% is inclusive, so if 20% of the leaves remain the first output should be `true`.
- You may find the `sprintf()` function helpful.

Drill Problem #4

Function Name: powerOfTwo

Inputs:

1. (*double*) A positive whole number

Outputs:

1. (*logical*) A logical value specifying if the input is a power of two

Function Description:

For all our computations, we have been using the base-10 decimal number system. However, computers (and MATLAB!) store numbers using the base-2 system, called binary. A binary number consists of multiple bits of either `0` or `1`. Each bit `n` represents the decimal value 2^n, and a binary number can be converted to decimal by adding all the powers of two corresponding to each bit that is a `1`.

For example, the decimal number 214 in binary is 11010110.

128	64	32	16	8	4	2	1	Value
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^n
1	1	0	1	0	1	1	0	n Bits

11010110 (binary) = $2^7 + 2^6 + 2^4 + 2^2 + 2^1$ = 128 + 64 + 16 + 4 + 2 = 214 (decimal)

Here is a table of the decimal numbers 0 – 7 and the equivalent binary numbers

7	6	5	4	3	2	1	0	Decimal
111	110	101	100	011	010	001	000	Binary

Using logical indexing, write a MATLAB function that calculates if a decimal number is a power of two. Return a logical (`true` or `false`) value specifying whether or not the decimal number is a power of two.

Notes:

- Since it has not been taught in class, you should not use iteration to solve this problem.
- You may not use the `log()`, `log2()`, or `dec2binvec()` functions.
- You will only be provided positive whole numbers. There are no negative powers of two!

Hints:

- The `dec2bin()` function will be useful. Pay attention to the output!

Drill Problem #5

Function Name: deflateGate

Inputs:

1. (*double*) An MxN array representing the first quarter
2. (*double*) An MxN array representing the second quarter
3. (*double*) An MxN array representing the third quarter
4. (*double*) An MxN array representing the fourth quarter

Outputs:

1. (*double*) An MxN array of the average pressure values
2. (*logical*) An MxN array indicating the games with broken pressure gauges

Function Description:

Due to recent predicaments in the National Football League (NFL), the NFL has decided to record the pressure of their footballs after every quarter in a game. Now that the preseason is over and the real season is about to begin, they have "recruited" you to write a MATLAB function to take the pressures recorded at the end of each quarter and find the average pressure of each football in every game. The pressure in a football will change slightly over the course of a game due to various weather conditions, usage, and other nefarious activities. Footballs that read the same pressure between any two consecutive quarters have fallen victim to suspicious activities, and we need you to identify them in your final analysis.

You are given four input MxN arrays, representing one of the four quarters in a football game. Each array will contain all of the pressure readings in a specific quarter of the footballs for the different games. Your job is to average all four values of each position in the arrays and output this averaged array for the first output. Any values that remain the same in a given index between two consecutively quartered arrays should not have an average displayed, since that means nefarious activity has occurred and the pressure gauge is broken; instead the final average array should have a zero at each of those indices. You should also output a second array, this one of class `'logical'`, which has a `true` at each index where the broken pressure gauges are being used in the football game.

Notes:

- The four input arrays are guaranteed to have the same dimensions.
- Included in the test cases are `'.mat'` files. Typing `load('<file_name>.mat')` into the Command Window, or double-clicking the `'.mat'` file in the Current Directory, will load any variables saved in the file to the Workspace.
- Do NOT load any of the `'.mat'` files inside your function.

Drill Problem #6

Function Name: camelCase

Inputs:

1. (*char)* A string of some phrase to be converted to camel case

Outputs:

1. (*char*) The phrase converted to camel case

Function Description:

In this day and age of fancy technology and radical lingo, you decide that you're going to text and email phrases in camel case to be as efficient as possible. Who needs spaces and vowels anyways?

Write a MATLAB function called camelCase that will take a phrase of characters and convert it to camel case. The algorithm to convert a string to camel case for this problem is as follows:

1. Capitalize the first letter of every individual word, except for the very first letter of the phrase. Lowercase every other letter.
2. Remove all spaces and extraneous special characters from the phrase.
3. Remove all vowels from the phrase, excluding those that are the first letter of a word.

For example, for the input:

```
'MATLAB is awesome'
```

The output would appear as:

```
'mtlbIsAwsm'
```

While the input

```
'I love MATLAB'
```

Would output the string:

```
'iLvMtlb'
```

Notes:

- This problem must be completed using logical indexing.
- The letter `'y'` should not be considered a vowel for this function.

Hints:

- The order in which you manipulate the input phrase will need to be taken into consideration.