Function Name: `cookieThrow`

Inputs:

1. (*double*) A 1xM vector of initial velocities
2. (double) A 1xM vector accelerations
3. (double) A 1xM vector of times

Outputs:

1. (*double*) A 1xM vector of the final velocities

Topics: (*vector math*)

Background:

You and your friend are throwing cookies at each other trying to catch them in your mouths. You record data about cookies upon release and want to know the speed of the cookies after they travel a certain amount of time. You decide to code a MATLAB function to make the process faster!

Function Description:

Given vectors of initial velocities (`V`$_o$), accelerations (`a`), and times (`t`), calculate a vector of final velocities (`V`$_f$). Use the following equation to solve:

$$V_f = V_o + at$$

Example:

```
velocity = [2, 3, 4, 5]
acceleration = [1, 2, 3, 4]
time = [2, 1, 10, 2]

updatedVelocity = cookieThrow(velocity, acceleration, time)

updatedVelocity = [4, 5, 34, 13]
```

Function Name: `unscrambledEggs`

Inputs:

1. (*double*) 1XN vector of the order in which to unscramble the recipe
2. (*char*) 1X(2N) vector representing the scrambled recipe

Outputs:

1. (*char*) 1X(2N) vector representing the unscrambled recipe

Topics: (*LHS/RHS indexing*)

Background:

You've been MATLABing all day, and man, are you hungry. You're feeling breakfast for dinner, so you turn to your trusty cookbook to find the recipe for your favorite breakfast meal: scrambled eggs! But it looks like you're out of luck! Your roommate pranked you and replaced your cookbook with another. All the words on the recipe are jumbled, but they seem to follow a pattern. With your stomach growling, you turn to MATLAB for a quick save so you don't *die* of hunger tonight.

Function Description:

Given a character vector that represents your cookbook recipe, use the double vector to unscramble your recipe. The double vector represents a pattern that will unjumble the characters and arrange them in the correct order. However, the pattern given is only half of the length of the entire recipe. In order to successfully unjumble the recipe, you must apply the pattern to the character vector twice: once to the first half, and once to the second half. After unscrambling, output the corrected recipe.

Example:

```
pattern = [7, 4, 1, 3, 10, 13, 12, 8, 11, 5, 9, 2, 6]
scrambled = 'w  okSHMeTa oagmrEscdgb el'

unscrambled = unscrambledEggs(pattern, scrambled)
unscrambled → 'How To Make Scrambled Eggs'
```

Notes:

- The character vector will always be exactly twice the length of the pattern.
- The *same* pattern must be applied to the first and second half of the recipe.

Hints:

- Think about how you might split the recipe in half to make it easier to apply the pattern.

Function Name: `grilledCheese`

Inputs:

1. (*double*) 1XM vector of your scores
2. (*double*) 1XN vector of your friend's scores
3. (*double*) 1XK vector of your second friend's scores

Outputs:

1. (*double*) 1X3 vector of score statistics

Topics: (*mean/median/mode*), (*sorting*)

Banned Functions:

```
mean, median
```

Background:

There is only ONE correct way to make grilled cheese. Luckily, the world's greatest comedian, Tom Middleditch, can teach you how to perfect the art of the grilled cheese sandwich. A-boopity-boop-boop!

Function Description:

You and two friends make grilled cheese, and receive scores from various judges. Your scores are represented by the values in the first input vector. Your first friend's scores are represented by the values in the second input vector. Your second friend's scores are represented by the values in the third input vector.

Find the **average** of your scores (rounded to one decimal place), the **median** of your first friend's scores, and the **mode** of your second friend's scores. The `mean()` and `median()` functions are banned, so you will have to figure out how to calculate these values without the aid of those functions. Finally, combine these three values into one vector, and sort this vector in **descending** order.

Example:

```
myScores = [10, 9, 10]
friend1 = [1, 0, 10000, 0, -1]
friend2 = [1, 2, 1, 1000]

scoreStats = grilledCheese(myScores, friend1, friend2)
scoreStats → [9.7, 1, 0]
```

Notes:

- The length of the second input is guaranteed to be odd
- One value is guaranteed to appear more than the others in the third input

Hints:

- To sort in descending order, remember that the `sort()` function requires a second input

Function Name: `mealPrep()`

Inputs:

1. (*char*) A 1 x N char vector containing a message

Outputs:

1. (char) a 1X N char vector of the decoded sentence

Topics: *(char vectors),(LHS/ RHS indexing), (colon operator), (strfind/strrep), (casting)*

Background:

You're a broke and hungry college student, but your grandma called you and told you about the concept of meal prepping your food for the week to save both time and money! She texted you her favorite cooking sayings and advice over her old cracked iPhone 3, but the message came out jumbled, and you're unable to interpret it. Thanks, Grandma. Luckily, she had done the same thing to your mom too who told you the steps to decode the message using MATLAB!

Function Description:

Given a character vector containing a message, follow these steps in the correct order to decipher it:

1. Starting with the first index of the char vector, shift every **third** index **up** 2 ASCII values (for example,`'q' → 's'`)
2. Replace all underscores `'_'` with a single space
3. Capitalize the first letter of every word in the sentence

Example:

```
mixed = 'qalr_tfe_uatcr_`efmre]boglilg_gt'

fixed = mealPrep(mixed)
fixed → 'Salt The Water Before Boiling It'
```

Notes:

- Make sure to capitalize the first letter of every word, including the first word (step 3)
- Words will be separated by a single underscore
- The original statement may contain special characters, but shifting should remove all special characters (except underscores), and the final output should not contain any

Hints:

- You may need to **cast** the char vector to a **different data type** to perform a mathematical "shift" for step 1

- The letters at the beginning of a word are always one index after a space except for the very first character of the string
- The colon operator may be useful

Function Name: `blender( )`

Inputs:

1. (*char*) An incorrectly formatted list of ingredients
2. (*char*) The single ingredient you are looking for

Outputs:

1. (*char*) A sentence outputting how much of the specified ingredient you need

Topics: (*char strings*), (*sprintf*), (*LHS/RHS indexing*)

Background:

You're cleaning out your kitchen and you come across an old cookbook with family recipes. Unfortunately, the ingredient lists for each recipe are full of typos, and you can't tell how much of each ingredient you need. However, it seems like someone before you has left notes telling you how to decipher the recipes, so you turn to MATLAB to figure out what you need!

Function Description:

Given a 1xN char vector with a jumbled list of ingredients and a 1xM char vector of a single ingredient, use the following instructions to determine how much of the ingredient you need:

1. You realize the list is written backwards, so first you must reverse the entire list such that the last letter becomes the first letter, the second-to-last letter becomes the second letter, and so on.
2. Next, you notice that there are random exclamation marks throughout the list. Delete every exclamation mark that appears.
3. Now, find the starting index of the desired ingredient (input 2) in the list. The starting index of the ingredient in the list represents the amount of that ingredient that you need.
4. Lastly, output a sentence summarizing what you need with the following format:

   ```
   'I need to buy <amount> <specified ingredient>(s).'
   ```

   Be sure to print the specified ingredient exactly as it was given to you (all lowercase).

Example:

```
list = '!!!IwIk!N!o!M!e!L!'
ingredient = 'kiwi'
[str] = blender(list, ingredient)
str → 'I need to buy 6 kiwi(s).'
```

Notes:

- The incorrectly formatted list may contain both upper and lowercase letters. The list will not contain any spaces or special characters other than '!'.
- The ingredient you are searching for (input 2) will always be one word. It will always be lowercase and will not contain any spaces or special characters.
- The ingredient you are searching for may contain both upper and lowercase letters when it appears in the list.
- The ingredient you are searching for is guaranteed to appear only once in the list.
- When outputting the final sentence, print the ingredient exactly as it was given to you.

Hints:

- Consider ways you can account for case-sensitivity (i.e. mAtLaB is not the same as MATLAB)

Extra Credit

Function Name: `secretFormula`

Inputs:

1. (*double*) 1xM number-vector containing a number represented in a different base
2. (*double*) The base the number-vector is represented in
3. (*double*) The cipher shift amount
4. (*char*) 1xN encrypted string
5. (*double*) 1xK quantity vector

Outputs:

1. (*char*) A sentence describing how many cups of sugar are in the secret formula

Banned Functions:

`dec2base, base2dec, circshift, shiftdim, fix`

Topics: (*cryptography—caesar cipher*), (*vectors*), (*base conversion*)

Background:

Legend has it that, long ago, a renowned chef prodigy developed a cake so delicious, reality itself rejoiced in its existence. However, as history proves again and again, such greatness can never last. The formula was lost to a terrible storm, which the prodigy encountered on his way to deliver a slice of that beloved cake to a wealthy Nigerian prince (some say the chef was offered a great fortune for that slice!). Lost over countless millennia with the rise and fall of the seas, the elusive recipe found itself on a shallow bank at the edge of a large archaeological site. The passage of time was not kind to that formula...

By day, you are a respected archaeologist in charge of the local dig site; by night, you indulge in the most popular baking shows on Netflix. Tonight's show: The Great British Baking Show! As you settle down to start the show, you feel your phone ring. It's your buddy Jimbo.

> "Boss, you've got to come back to the site! You won't believe what we found!"
>
> "What could possibly be so important to interrupt my baking show?"
>
> "I'm sorry boss, but you've got to come see this. Did you ever hear the tragedy of the baker and his cake?"
>
> "No, what are you talking about??"
>
> "OMG how haven't you heard it before? Legend has it that, long ago, a…"
>
> … *(10 minutes later)* ...
>
> "So boss whaddaya say? Wanna come take a peek?"
>
> "I'll be there in 30."

Enchanted by the prospect of baking the ultimate treat, you grab your briefcase and coat, and yeet yourself to the dig site. On your drive there, your mind flares with possibility. What are the main ingredients and optimal ratios? Do I need a metal whisk or will a plastic one do? Why does

Jimbo call me boss so much? You take a deep breath and keep driving. When you arrive, you find an empty spot near the site entrance and rush over to what seems to be a makeshift campsite a good bit south of the western hub. With an energized stride, you quickly make it to the site. You walk up to Jimbo and the crew, who appear to be dumb-founded by the ancient slip of paper in front of them. They're huddled around a table discussing who-knows-what when Jimbo notices your approach.

"Boss! We've been at it for a while now boss, but we can't seem to figure it out!"

"Pass it here Jimbo."

Jimbo passes you what you immediately recognize as the legendary recipe. Finally, you're face to face with eons of lost perfection! The only problem is, *it's all encrypted*. Luckily you were a cryptography junkie back in secondary school! After a few minutes, you manage to crack most of the encrypted formula, but you've reached perhaps the most difficult encryption you've ever encountered. You rise to the challenge. The time has come. You crack your knuckles, set your briefcase on the table, take out your computer, pull up MATLAB, and get to work.

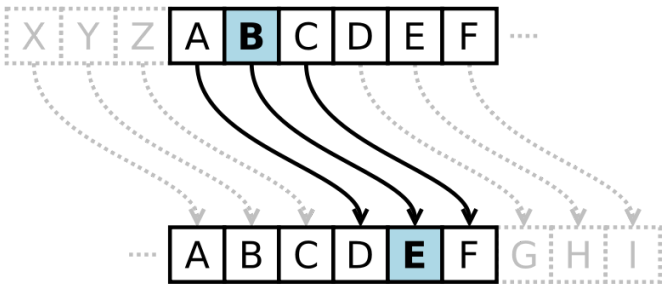
Function Description:

You are given an encrypted string (*input 4*) and several clues as to how to decrypt it. To decrypt your encrypted string, begin by converting the number-vector (*input 1*) to base 10. Each element of the number-vector corresponds to a single digit of the number in a given base (*input 2*). An example of the format of the number-vector is shown below.

```
base = 2
numVec = [1,0,1,1,0] % This would normally be written as 10110,
                     % and in base 10 this equals 22.
base10Num → 22
```

After converting the number-vector to a number in base 10, index the first `base10Num` values from the encrypted string. In this example, if your encrypted string was 1xP, P ≥ 22, you'd index out the values from positions 1 to 22 to form a sub-string that is 22 characters long.

Take this sub-string and perform a caesar shift using the shift amount (*input 3*) to get the decrypted string. A positive shift amount shifts the alphabet to the right, and a negative shift amount shifts the alphabet to the left. The alphabet wraps around. A visual representation of a caesar shift (shift value of 3) is shown below.



The decrypted string contains characters whose ASCII values correspond to a particular index of the quantity vector (*input 5*). In order to get the number of cups of sugar in your secret formula, you must sum the values at those indices together. For example, if:

```
encryptedStr = 'ipxez'
```

and:

```
decryptedStr = 'howdy'
```

then:

```
inds → [8 15 23 4 25] + 96      % 96 is 1 less than double('a')
```

You would then index the quantity vector with these indices, and sum the values indexed from the quantity vector together for the final amount. Output the number of cups needed in a formatted string, following the template below:

```
'The secret formula needs <number of cups> cups of sugar!'
```

Example:

```
>> numVec = [1 1 2]; % this is 14 in base 3
>> base = 3;
>> shift = -3;
>> encStr = 'etqbxjybpqqbxjbfbrzyfrntyuktnlfuekltfne'
>> quantVec = [8 2 3 5 6 3 1 5 2 5 7 4 2 2 4 5 8 9 9 2 ...
               2 4 5 6 7 3 5 6 2 4 5 6 8 8 1 9 9 8 3 1 ...
               1 3 8 5 2 8 3 6 4 8 2 7 7 2 8 2 2 8 2 9 ...
               7 3 6 4 7 3 8 4 7 2 8 1 9 1 8 3 7 2 6 1 ...
               8 3 6 4 6 4 8 2 9 1 7 3 6 4 9 1 8 3 7 4 ...
               2 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 3 5];
>> desc = secretFormula(numVec, base, shift, encStr, quantVec);

desc →
     'The secret formula needs 55 cups of sugar!'

% Below are intermediary values. They are only for your
reference
% you do not need to output them.
decStr →
     'hwteambestteam'
inds →
     [104 119 116 101 97 108 98 101 115 116 116 101 97 109]
```

Notes:

- You will not be asked to convert from above base 9 (inclusive).
- You will only be asked to convert positive integers to base 10.
- There will be no leading 0's in the number-vector.
- The shift value can be any integer, positive or negative.

- The encrypted string will only contain lowercase letters (i.e. no uppercase, special characters, or numbers).
- There are guaranteed to be enough elements in the quantity vector to index sans error.
- Remember, iteration and conditionals are banned!

Hints:

- Think about how you could use vector operations, `mod()`, and helper functions to solve this problem!
- Research the problem's topics and work things out on paper prior to implementing it in MATLAB.
- Start with small, simple examples and work your way up to a general solution.