Notice

This homework will require you to create functions that output spreadsheet files. These cannot be directly compared using `isequal`. However, we can compare them with a few lines of code!

```
>> functionName(in...);
>> functionName_soln(in...);
>> check = isequal(readcell('file.xlsx'), readcell('file_soln.xlsx'));
```

Happy Coding!
~Homework Team

Function Name: `breadDeadRedemption`

Inputs:

1. (*char*) Partial start to a file name

Outputs:

1. (*cell*) An Mx1 cell array containing all files starting with the partial name

Topics: (*dir*), *(structures)*

Background:

In exchange for becoming the next owner of the CS1371 Bakery, you agree to track down all of the different recipes, which are hidden somewhere in the wild, wild west, or more accurately, your current directory.

Function Description:

Given the starting portion of a file name, return a cell array of all files and directories (folders) in your current directory that start with that partial name. If no files/directories match the input, return a 1x1 cell containing `'No Files Found'`.

Example:

Current Folder: (displayed as seen in the Current Folder window in MATLAB)

grades.txt
gradebook.xlsx
GradeReplacementPolicy.txt
SampleGrades.txt

```
files = breadDeadRedemption('grade')

files → {'gradebook.xlsx'; 'grades.txt'}
```

Notes:

- File names are case sensitive
- Keep your cell array in the same order (ASCII-ordered) that `dir()` returns the files/directories

Function Name: `slapMyBread`

Inputs:

1. (*char*) The name of the excel file containing physics equations

File Outputs:

1. An excel file containing the solved equations

Topics: (*High Level I/O*), (*Utilizing Previous Code*)

Background:

You're chilling at the CULC when suddenly you receive an email from Dr. Greco asking you to come to his office—immediately. Panicked, you run over expecting the worst; to your surprise however, he commends your perfect physics homework from a couple of weeks ago and asks for your help. In a new research project, Dr. Greco projects that it is perhaps more efficient to bake a loaf of bread by slapping it with your hand at a high velocity and thereby transferring all kinetic energy into thermal energy in an inelastic collision. In order to test out this theory, he needs lots of test values calculated by you to find the kinetic energy, mass, or velocity for various combinations to optimize his system.

Function Description:

You are trying to solve the following kinetic energy equation:

$$K = \frac{1}{2}mv^2$$

You are given the name of an excel file with three columns of data. The columns of the file correspond to K, m, and v respectively. In each row, two columns contain known variable values and one contains an unknown variable represented by `'K'`, `'m'`, or `'v'` respectively. Solve for the unknown variable in each row and write the results to a new cell array with two columns. The first column of the new excel file should contain the variable you solved for in that row, and the second column should contain its value. Name the new excel file using the following format: `<original filename>_solved.xlsx`.

Example:

`breadData.xlsx`:

K	5	2
50	4	v
70	m	9

```
slapMyBread('breadData.xslx')
```

`breadData_solved.xlsx`:

K	10
v	5
m	1.728

Notes:

- You are guaranteed that the excel data will be rectangular (i.e. there will never be any empty cells within the data).
- Each row is guaranteed to have only one unknown variable
- Each number calculated must be rounded to the thousandths place.

Hints:

- You may want to call your `physicsHW` function from Homework 8. If you do, you ***must*** submit `physicsHW.m`, or paste it into your `slapMyBread.m` file as a helper function.

Function Name: `getThisBread`

Inputs:

1. (*char*) The filename of a text file containing your shopping list
2. (*char*) The filename of an Excel file containing a grocery store inventor

Outputs:

1. *(char*) A string stating the total cost of your grocery store run.

Topics: (*Low Level Reading*), (*Lookup Table*)

Background:

It is time to get this 🅱️read.

Function Description:

You are given a grocery list text file with each line in the form:

`'<item>,<number of items>'`

and an Excel file that contains an inventory for the grocery store you are visiting. The first column in the inventory has the number of items in stock, the second column will have the item name, and the third column has the price per unit (in dollars). Write a function that calculates the total cost of your grocery store run and outputs the string:

`'My total will be $<total>.'`

Example:

`inventory.xlsx` →

13	Butter Bread	2.08
5	Whole Wheat Bread	4.02
36	Honey Wheat Bread	0.72

`list.txt` →

```
1| whole wheat bread,1
2| BUTTER BREAD,12
```

```
out = getThisBread('list.txt', 'inventory.xlsx')
out → 'My total will be $28.98.'
```

Notes:

- Matching grocery items should be case insensitive.
- You are guaranteed to find everything you need and the store will always have enough.
- You do not have to update the store's inventory
- Round your price to 2 decimal places.

Function Name: `sliceAndDice`

Inputs:

1. (*char*) A string representing a partial filename

Outputs:

1. (*cell*) An MxN cell array of strings

Topics: (*dir*), (*cell creation*), (*string tokenization*)

Background:

Your bread lab TA just sent you the data for last week's session. Why was the lab about bread? Why does the data you collected look weird and terrible? Who knows?

Function Description:

You are given a partial name of a delimited text file; however, each line has a different delimiter. Each line will end in the correct delimiter for that specific line. Take the text file and turn it into an excel file with the same name.

Example:

`alpha.txt` →

```
1| Symbol_Sound_
2| a,aye,
3| bIbeeI
```

```
sliceAndDice('alph');
```

`alpha.xlsx` →

Symbol	Sound
a	aye
b	bee

Notes:

- The last character of every line will be that lines delimiter.
- There will be only one file in the directory which contains the partial filename.
- All data will be of type char.

Hints:

- `contains()` may be useful to find the correct file.

Function Name: `breadStats`

Inputs:

1. (*char*) Name of an Excel file, including the extension .xlsx (<filename>.xlsx)
2. (*char*) 1xN cell array with the names of bread that are on sale

File Outputs:

1. An Excel file named <filename>_calculated.xlsx of the fixed stats

Topics: (*headers*), (*cell operations*)

Background:

As a stressed Tech student, you decided to eat some cheap bread for the rest of the week. Since you have no time to work out during hell week, the bread with lowest calories will also be better. To consider all factors and get the best deal, you are using MATLAB to help you get the best bread!

Function Description:

You will always be given the following information as the headers for each column: `'Bread'`, `'Carbs'`, `'Proteins'`, `'Fats'`, and `'Bread_Price'`, but keep in mind they will be in random order. Process the results using the following rules:

- Delete the types of bread that are not on the on sale list(the second input)
- Since you only have the weight (in gram) of three nutrients: carbohydrate, protein, and fat, you need to calculate the total calories of the bread.
    - Carbohydrates have 4 calories per gram
    - Proteins have 4 calories per gram
    - Fats have 9 calories per gram
    - Make a new column for this with the header `'Calories'` and store the calculated calories of each kind of bread in it.
- Rank the bread! The higher points a bread has, the better it is. The points are calculated based on price and calories, each scored out of a maximum of 10 points.
    - Price: Every 2 dollars, the number of points the bread can earn goes down by 1.
        - For example: price of 0-1.99 dollars earns 10 points, price of 2-3.99 dollars earns 9 points etc.
    - Calories: Every 50 calories, the number of points the bread can earn goes down by 1.
        - For example: 0-49 earns 10 points. 50-99 earns 9 points etc.
    - Add these price and calories score together and store them in a new column called `'Total_Points'`.
- Sort the bread stats by their total point in ascending order
- Sort the table again with their headers in alphabetical order.
- Output the top three types of bread with the highest total points with headers in `'<filename>_calculated.xlsx'`

Example:

`stats.xlsx →`

Bread	Fats	Carbs	Proteins	Bread_Price
White bread	1	45	6	8.00
Croissant	3	20	3	6.00
Bagel	7	30	9	3.00
Banana bread	8	10	14	8.00
Bread stick	9	0	9	3.00

```
onsale = {'Croissant', 'Bread stick', 'White bread'};

breadStats('stats.xlsx', onsale)→
```

`stats_calculated.xlsx →`

Bread	Bread_Price	Calories	Carbs	Fats	Proteins	Total_Points
White bread	8.00	213	45	1	6	12
Croissant	6.00	119	20	3	3	15
Bread stick	3.00	117	0	9	9	17

Notes:

- There will be no tie among the top 3 types of bread.
- The point will always be positive.
- There will always be at least 3 bread on sale.
- Other than the `'Bread'` column and the headers, all cells will contain data of type double.