Exercise 24 - Anagram Checker

Using functions to abstract the logic away from the rest of your code makes it easier to read and easier to maintain.

Create a program that compares two strings and determines if the two strings are anagrams. The program should prompt for both input strings and display the output as shown in the example that follows.

Example Output

```
Enter two strings and I'll tell you if they are anagrams:
Enter the first string: note
Enter the second string: tone
"note" and "tone" are anagrams.
```

Constraints

- Implement the program using a function called `isAnagram`, which takes in two words as its arguments and returns true or false.

Challenge

- Complete this program without using built-in language features. Use selection or repetition logic instead and develop your own algorithm.

Exercise 25 - Password Strength Indicator

Functions help you abstract away complex operations, but they also help you build reusable components.

Create a program that determines the complexity of a given password based on these rules:

- A very weak password contains only numbers and is fewer than eight characters.
- A weak password contains only letters and is fewer than eight characters.
- A strong password contains letters and at least one number and is at least eight characters.
- A very strong password contains letters, numbers, and special characters and is at least eight characters.

If a password does not meet any of these rules, then report it as a password of unknown strength.

Example Output

```
The password '12345' is a very weak password.
The password 'abcdef' is a weak password.
The password 'abc123xyz' is a strong password.
The password '1337h@xor!' is a very strong password.
```

Constraints

- Create a `passwordValidator` function that takes in the password as its argument and returns a numeric value you can evaluate to determine the password strength.

- Use a single output statement.

Challenge

- Create a GUI application or web application that displays graphical feedback as well as text feedback in real time. As someone enters a password, determine its strength and display the result

Exercise 26 - Months to Pay Off a Credit Card

It can take a lot longer to pay off your credit card balance than you might realize. And the formula for figuring that out isn't pretty. Hiding the formula's complexity with a function can help you keep your code organized.

Write a program that will help you determine how many months it will take to pay off a credit card balance. The program should ask the user to enter the balance of a credit card, the APR of the card, and their monthly payment. The program should then return the number of months needed (rounded up to the next integer value).

The formula for this is

```
n = -(1/30) * log(1 + b/p * (1 - (1 + i)^30)) / log(1 + i)
```

where

- n is the number of months.
- i is the daily rate (APR divided by 365).
- b is the balance.
- p is the monthly payment.

Example Output

```
What is your balance? 5000
What is the APR on the card (as a percent)? 12
What is the monthly payment you can make? 100
It will take you 70 months to pay off this card.
```

Constraints

- Prompt for the card's APR. Do the division internally.
- Prompt for the APR as a percentage, not a decimal.
- Use a class called `PaymentCalculator` with a public method called `calculateMonthsUntilPaidOff`, which takes no parameters and returns the number of months.
- Round fractions of a cent up to the next cent when displaying information to the user.

Challenge

- Rework the formula so the program can accept the number of months as an input and compute the monthly payment.
- Create a version of the program that lets the user choose whether to figure out the number of months until payoff or the amount needed to pay per month.

Exercise 27 - Validating Inputs

Large functions aren't very usable or maintainable. It makes a lot of sense to break down the logic of a program into smaller functions that do one thing each. The program can then call these functions in sequence to perform the work.

Write a program that prompts for a first name, last name, employee ID, and ZIP code. Ensure that the input is valid according to these rules:

- The first name must be filled in.
- The last name must be filled in.
- The first and last names must be at least two characters long.
- An employee ID is in the format AA-1234. So, two letters, a hyphen, and four numbers.
- The ZIP code must be a number.

Display appropriate error messages on incorrect data.

Example Output

```
Enter the first name: J
Enter the last name:
Enter the ZIP code: ABCDE
Enter the employee ID: A12-1234
The first name must be at least 2 characters long.
The last name must be at least 2 characters long.
The last name must be filled in.
The employee ID must be in the format of AA-1234.
The zipcode must be a 5 digit number.
```

Or

```
Enter the first name: John
Enter the last name: Johnson
Enter the ZIP code: 55555
Enter the employee ID: TK-4321
There were no errors found.
```

Constraints

- Create a function for each validation rule. Then create a `validateInput` function that takes in all of the input data and invokes the specific validation functions.
- Use a single output statement to display the outputs.

Challenges

- Use regular expressions to validate the input.
- Implement this as a GUI application or web application that gives immediate feedback when the fields lose focus.
- Repeat the process if the input is not valid.

Exercise 28 - Adding Numbers

In previous programs, you asked the user for repeated input by writing the input statements multiple times. But it's more efficient to use loops to deal with repeated input.

Write a program that prompts the user for five numbers and computes the total of the numbers.

Example Output

```
Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: 4
Enter a number: 5
The total is 15.
```

Constraints

- The prompting must use repetition, such as a counted loop, not three separate prompts.

Challenges

- Modify the program to prompt for how many numbers to add, instead of hard-coding the value. Be sure you convert the input to a number before doing the comparison.
- Modify the program so that it only adds numbers and silently rejects non-numeric values. Count these invalid entries as attempts anyway. In other words, if the number of numbers to add is 5, your program should still prompt only five times.

Exercise 29 - Handling Bad Input

The rule of 72 is a quick method for estimating how long it will take to double your investment, by taking the number 72 and dividing it by the expected rate of return. It's a good tool that helps you figure out if the stock, bond, or savings account is right for you. It's also a good program to write to test for and prevent bad input because computers can't divide by zero. And instead of exiting the program when the user enters invalid input, you can just keep prompting for inputs until you get one that's valid.

Write a quick calculator that prompts for the rate of return on an investment and calculates how many years it will take to double your investment.

The formula is

```
years = 72 / r
```

where r is the stated rate of return.

Example Output

```
What is the rate of return? 0
Sorry. That's not a valid input.
What is the rate of return? ABC
```

```
Sorry. That's not a valid input.
What is the rate of return? 4
It will take 18 years to double your initial investment.
```

Constraints

- Don’t allow the user to enter 0.
- Don’t allow non-numeric values.
- Use a loop to trap bad input, so you can ensure that the user enters valid values.

Challenge

- Display a different error message when the user enters 0.

Exercise 30 - Multiplication Table

Create a program that generates a multiplication table for the numbers 1 through 12 (inclusive).

Example Output

```
   1    2    3    4    5    6    7    8    9   10   11   12
   2    4    6    8   10   12   14   16   18   20   22   24
   3    6    9   12   15   18   21   24   27   30   33   36
   4    8   12   16   20   24   28   32   36   40   44   48
   5   10   15   20   25   30   35   40   45   50   55   60
   6   12   18   24   30   36   42   48   54   60   66   72
   7   14   21   28   35   42   49   56   63   70   77   84
   8   16   24   32   40   48   56   64   72   80   88   96
   9   18   27   36   45   54   63   72   81   90   99  108
  10   20   30   40   50   60   70   80   90  100  110  120
  11   22   33   44   55   66   77   88   99  110  121  132
  12   24   36   48   60   72   84   96  108  120  132  144
```

Constraint

- Use a nested loop to complete this program.
- Align each column within the table without using the tab character.

Challenges

- Create a graphical program. Use a drop-down list to change the base number. Generate or update the table when the number is selected.

Exercise 31 - Karvonen Heart Rate

When you loop, you can control how much you increment the counter; you don’t always have to increment by one.

When getting into a fitness program, you may want to figure out your target heart rate so you don’t overexert yourself. The Karvonen heart rate formula is one method you can use to determine your rate. Create a program that prompts for your age and your resting heart rate. Use the Karvonen formula to determine the target heart

rate based on a range of intensities from 55% to 95%. Generate a table with the results as shown in the example output. The formula is

```
TargetHeartRate = (((220 – age) – restingHR) × intensity) + restingHR
```

Example Output

```
Resting Pulse: 65        Age: 22

Intensity    | Rate
-------------|--------
55%          | 138 bpm
60%          | 145 bpm
65%          | 151 bpm
:               :        (extra lines omitted)
85%          | 178 bpm
90%          | 185 bpm
95%          | 191 bpm
```

Constraints

- Don’t hard-code the percentages. Use a loop to increment the percentages from 55 to 95.
- Ensure that the heart rate and age are entered as numbers. Don’t allow the user to continue without entering valid inputs.
- Display the results in a tabular format.

Challenge

- Implement this as a GUI program that allows the user to use a slider control for the intensity, and update the interface in real time as the slider moves.

Exercise 32 - Guess the Number Game

Write a Guess the Number game that has three levels of difficulty. The first level of difficulty would be a number between 1 and 10. The second difficulty set would be between 1 and 100. The third difficulty set would be between 1 and 1000.

Prompt for the difficulty level, and then start the game. The computer picks a random number in that range and prompts the player to guess that number. Each time the player guesses, the computer should give the player a hint as to whether the number is too high or too low. The computer should also keep track of the number of guesses. Once the player guesses the correct number, the computer should present the number of guesses and ask if the player would like to play again.

Example Output

```
Let's play Guess the Number!

Enter the difficulty level (1, 2, or 3): 1
I have my number. What's your guess? 5
Too low. Guess again: 7
Too low. Guess again: 9
You got it in 3 guesses!
```

```
Do you wish to play again (Y/N)? y

Enter the difficulty level (1, 2, or 3): 3
I have my number. What's your guess? 500
Too low. Guess again: 750
Too high. Guess again: 600
Too low. Guess again: 700
Too low. Guess again: 725
Too high. Guess again: 715
Too high. Guess again: 710
Too high. Guess again: 705
Too high. Guess again: 701
Too low. Guess again: 702
You got it in 10 guesses!

Do you wish to play again (Y/N)? n
```

Constraints

- Don’t allow any non-numeric data entry.
- During the game, count non-numeric entries as wrong guesses.

Challenges

- Map the number of guesses taken to comments:
  - 1 guess: “You’re a mind reader!”
  - 2–4 guesses: “Most impressive.”
  - 3–6 guesses: “You can do better than that.”
  - 7 or more guesses: “Better luck next time.”
- Keep track of previous guesses and issue an alert that the user has already guessed that number. Count this as a wrong guess.
- Implement this as a graphical game with a grid of numbers. When a number is clicked or tapped, remove the number from the screen.

Exercise 33 - Magic 8 Ball

Arrays are great for storing possible responses from a program. If you combine that with a random number generator, you can pick a random entry from this list, which works great for games.

Create a Magic 8 Ball game that prompts for a question and then displays either "Yes," "No," "Maybe," or "Ask again later."

Example Output

```
What's your question?
> Will I be rich and famous?

Ask again later.
```

Constraint

- The value should be chosen randomly using a pseudo random number generator. Store the possible

choices in a list and select one at random.

Challenges

- Implement this as a GUI application.
- If available, use native device libraries to allow you to “shake” the 8 ball.

Exercise 34 - Employee List Removal

Sometimes you have to locate and remove an entry from a list based on some criteria. You may have a deck of cards and need to discard one so it’s no longer in play, or you may need to remove values from a list of valid inputs once they’ve been used. Storing the values in an array makes this process easier. Depending on your language, you may find it safer and more efficient to create a new array instead of modifying the existing one.

Create a small program that contains a list of employee names. Print out the list of names when the program runs the first time. Prompt for an employee name and remove that specific name from the list of names. Display the remaining employees, each on its own line.

Example Output

```
There are 5 employees:
John Smith
Jackie Jackson
Chris Jones
Amanda Cullen
Jeremy Goodwin

Enter an employee name to remove: Chris Jones

There are 4 employees:
John Smith
Jackie Jackson
Amanda Cullen
Jeremy Goodwin
```

Constraint

- Use an array or list to store the names.

Challenges

- If the user enters a name that’s not found, print out an error message stating that the name does not exist.
- Read in the list of employees from a file, with each employee on its own line.
- Write the output to the same file you read in.

Exercise 35 - Picking a Winner

Arrays don’t have to be hard-coded. You can take user input and store it in an array and then work with it.

Create a program that picks a winner for a contest or prize drawing. Prompt for names of contestants until the

user leaves the entry blank. Then randomly select a winner.

Example Output

```
Enter a name: Homer
Enter a name: Bart
Enter a name: Maggie
Enter a name: Lisa
Enter a name: Moe
Enter a name:
The winner is... Maggie.
```

Constraints

- Use a loop to capture user input into an array.
- Use a random number generator to pluck a value from the array.
- Don’t include a blank entry in the array.
- Some languages require that you define the length of the array ahead of time. You may need to find another data structure, like an ArrayList.

Challenges

- When a winner is chosen, remove the winner from the list of contestants and allow more winners to be chosen.
- Make a GUI program that shows the array of names being shuffled on the screen before a winner is chosen.
- Create a separate contest registration application. Use this program to pull in all registered users and pick a winner.

Exercise 36 - Computing Statistics

Statistics is important in our field. When measuring response times or rendering times, it’s helpful to collect data so you can easily spot abnormalities. For example, the standard deviation helps you determine which values are outliers and which values are within normal ranges.

Write a program that prompts for response times from a website in milliseconds. It should keep asking for values until the user enters “done.”

The program should print the average time (mean), the minimum time, the maximum time, and the population standard deviation.

Example Output

```
Enter a number: 100
Enter a number: 200
Enter a number: 1000
Enter a number: 300
Enter a number: done
Numbers: 100, 200, 1000, 300
The average is 400.0
The minimum is 100
The maximum is 1000
The standard deviation is 353.55
```

Constraints

- Create functions called `average`, `max`, `min`, and `std`, which take in a list of numbers and return the results.
- Use loops and arrays to perform the input and mathematical operations.
- Be sure to exclude the "done" entry from the inputs.
- Be sure to properly ignore any invalid inputs.
- Keep the input separate from the processing and the output.

Challenges

- Have the program read in numbers from an external file instead of prompting for the values.

Exercise 37 - Password Generator

Coming up with a password that meets specific requirements is something your computer can do. And it will give you practice using random number generators in conjunction with a list of known values.

Create a program that generates a secure password. Prompt the user for the minimum length, the number of special characters, and the number of numbers. Then generate a password for the user using those inputs.

Example Output

```
What's the minimum length? 8
How many special characters? 2
How many numbers? 2
Your password is aurn2$1s#
```

Constraints

- Use lists to store the characters you'll use to generate the passwords.
- Ensure that the generated password is random.
- Ensure that there are at least as many letters are there are special characters and number.

Challenges

- Randomly convert vowels to numbers, such as 3 for E and 4 for A.
- Have the program present a few options rather than a single result.
- Place the password on the user's clipboard when generated.

Exercise 38 - Filtering Values

Sometimes input you collect will need to be filtered down. Data structures and loops can make this process easier.

Create a program that prompts for a list of numbers, separated by spaces. Have the program print out a new list containing only the even numbers.

Example Output

```
Enter a list of numbers, separated by spaces: 1 2 3 4 5 6 7 8
The even numbers are 2 4 6 8.
```

Constraints

- Convert the input to an array. Many languages can easily convert strings to arrays with a built-in function that splits apart a string based on a specified delimiter.
- Write your own algorithm—don't rely on the language's built-in filter or similar enumeration feature.
- Use a function called `filterEvenNumbers` to encapsulate the logic for this. The function takes in the original array and returns the new array.

Challenge

- Instead of prompting for numbers, read in lines from any text file and print out only the even-numbered lines.

Exercise 39 - Sorting Records

When you're looking at results, you'll want to be able to sort them so you can find what you're looking for quickly or do some quick visual comparisons.

Given the following data set, create a program that sorts all employees by last name and prints them to the screen in a tabular format.

First Name	Last Name	Position	Separation Date
John	Johnson	Manager	2016-12-31
Tou	Xiong	Software Engineer	2016-10-05
Michaela	Michaelson	District Manager	2015-12-19
Jake	Jacobson	Programmer	
Jacquelyn	Jackson	DBA	
Sally	Webber	Web Developer	2015-12-18

Example Output

```
Name               | Position          | Separation Date
-------------------|-------------------|----------------
Jacquelyn Jackson  | DBA               |
Jake Jacobson      | Programmer        |
John Johnson       | Manager           | 2016-12-31
Michaela Michaelson | District Manager  | 2015-12-19
Sally Weber        | Web Developer     | 2015-12-18
Tou Xiong          | Software Engineer | 2016-10-05
```

Constraint

- Store the data using a list of maps.

Challenges

- Prompt for how the records should be sorted. Allow sorting by separation date, position, or last name.
- Use a database such as MySQL, or a key-value store such as Redis, to store the employee records. Retrieve the records from the data store.

Exercise 40 - Filtering Records

Sorting records is helpful, but sometimes you need to filter down the results to find or display only what you're looking for.

Given the following data set create a program that lets a user locate all records that match the search string by comparing the search string to the first or last name field.

First Name	Last Name	Position	Separation Date
John	Johnson	Manager	2016-12-31
Tou	Xiong	Software Engineer	2016-10-05
Michaela	Michaelson	District Manager	2015-12-19
Jake	Jacobson	Programmer	
Jacquelyn	Jackson	DBA	
Sally	Webber	Web Developer	2015-12-18

Example Output

```
Enter a search string: Jac

Results:

Name                | Position           | Separation Date
--------------------|--------------------|----------------
Jacquelyn Jackson   | DBA                |
Jake Jacobson       | Programmer         |
```

Constraint

- Store the data using a list of maps.

Challenges

- Make the search case insensitive.
- Add the option to search by position.
- Add the option to find all employees where their separation date is six months ago or more.
- Read in the data from a file.