# The Code Challenge Book

## How to Ace the Coding Bootcamp Technical Interview
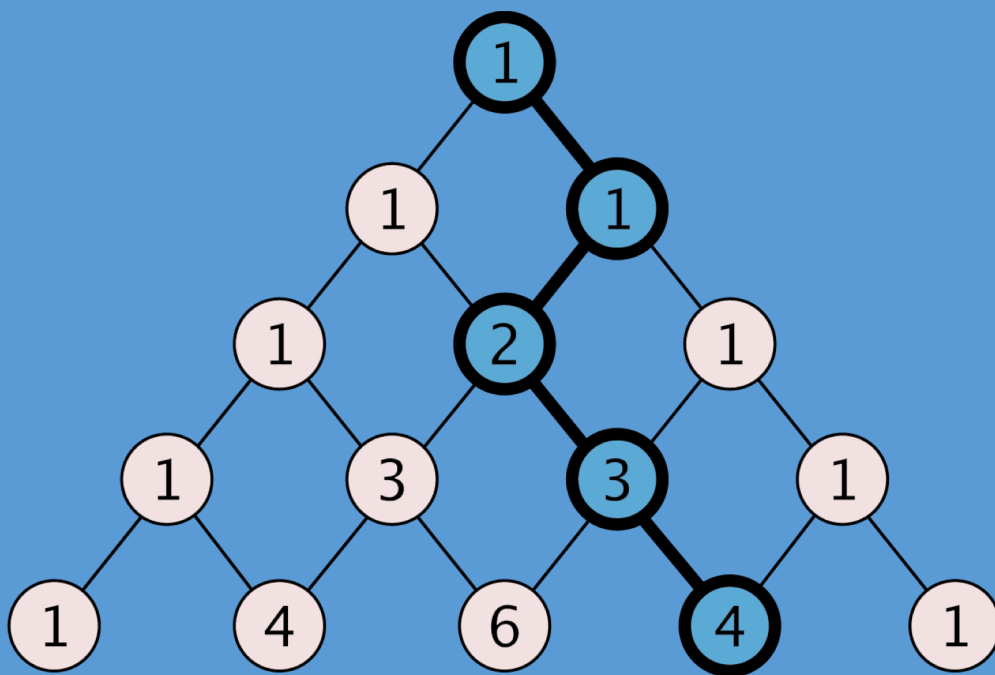
*By*

Daniel Borowski
Coderbyte

**TABLE OF CONTENTS**

PART 1
# BACKGROUND

---

*"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."*

*- Bill Gates*

## 1.1.    What is a coding bootcamp?

A coding bootcamp is an intensive training program that teaches students computer programming over the span of several weeks. The goal of these bootcamps is to condense the concepts taught in a standard college computer science program into a crash course that focuses explicitly on the *practical* skills and knowledge they offer. That way, students are prepared for software engineering jobs immediately after they finish the program. There are close to one hundred coding bootcamps in the United States alone that focus on teaching software development, data science, and mobile application development.[1] And it's not just the U.S. -- bootcamps are sprouting up all over the world, in countries like Portugal, Poland, France, and the U.K.[2] You can read more about bootcamps here: http://bit.ly/2eb7Xzf

During the bootcamp, students follow a rigorous schedule that typically runs over the course of 8-12 weeks.  It includes learning to code in a specific language, learning new software tools and methodology, studying algorithms, creating projects, working in groups, and other activities that are designed to prepare the student for the world of software engineering. On average, these bootcamps cost about $12,000 for the whole experience.[3]  Their steep tuition costs are justified by the promise that graduates will be able to find exciting jobs with high salaries once they're done. It can be a great option for someone with little to no coding experience who is looking to broaden their skills and transition into a new job. Once someone decides that this is the route they want to take, the first step is to complete an application online. One of the first challenges that person (you!) will face is **the coding challenge**.

## 1.2.    What is the purpose of a coding challenge?

When applying for a software engineering position at a company, the employer is typically going to do more than look at your resume and schedule a short interview. They may ask you to solve a few technical challenges, either online or on a whiteboard. This is done in order to:

1.  Determine how well you can code.
2.  Gauge your problem-solving abilities when faced with a novel challenge.
3.  Figure out if you can work through a difficult problem under a time limit.
4.  See how you handle a situation if you are stuck or confused.

Coding bootcamps follow the same model and typically ask their applicants to solve a few coding challenges at some point in the interview process. Below are a couple of sample bootcamp coding challenges:

---

[1] Course Report Market Sizing Report 2016: http://bit.ly/296nHFr
[2] Quora question about international coding bootcamps: http://bit.ly/2cRX0Em
[3] Course Report Market Sizing Report 2016: http://bit.ly/296nHFr

*You have been given a list of words. Write a program that will take this list and return a new list with all duplicate words removed.*

*You have been given a number. Write a program that will determine if this number is a prime number or not.*

During the interview, you will most likely have the option of asking questions to clarify your understanding of the problem. You will then be asked to give a high-level explanation of how you will work through the problem followed by writing a solution on the whiteboard or in a Word document. Afterwards, you will most likely discuss your code with the interviewer who may ask questions like why you wrote it in such a way, whether it will work for a given input, or how it might be made more efficient.

In the end, you will (hopefully) have written a program that does what it's supposed to and the interviewer will (hopefully) have gained an understanding of how you think when presented with a challenging problem, how you react to any issues that may arise, and whether you are adept at explaining your thought processes.

### 1.3.    Coding bootcamp interviews vs. job interviews

As mentioned, most coding bootcamps follow the same model for interviews as companies looking to recruitcandidates for coding jobs. These positions usually go by one of the following names:

- Software Engineer
- Software Developer
- Front-end Developer/Engineer
- Back-end Developer/Engineer
- Web Developer
- Programmer
- (Insert programming language here) developer

While the interview models may be similar, the *questions* asked in bootcamp interviews are going to be very different than those asked in interviews for engineering positions at companies like Google, for instance. When applying to a coding job at a company, the questions will typically be more complex. They may require knowledge in several domains of computer science, including data structures, algorithms, systems design, mathematical optimization, and statistics. They may also require much more code to actually solve the problem.

Below are some examples of challenges you may encounter at large technology companies like Google or Microsoft:

*Find the largest common subtree within two binary search trees. Solve this problem in linear time if you can.*

*How would you sort a 1 terabyte file of people's names if your RAM is limited to 4 gigabytes? What data structures would you use?*

You may be asking yourself: does the ability to solve these challenges mean someone is a good programmer? What if they can solve the problems but don't work great under the pressure of a time constraint? What if they just studied the solutions to a hundred coding challenges a few days before the interview but couldn't actually solve them on their own? Some members of the programming community are very critical of the "the technical interview," and there are plenty of articles, blog posts, and sites online that explain why these types of challenges may be flawed for determing whether someone is actually a good programmer.[1] Feel free to read the resources listed in the footnote on this topic, but know that as of now, you will most likely encounter coding challenges when applying to coding bootcamps and companies for engineering positions.

### 1.4. How are coding challenges related to engineering problems?

Coding challenges are meant to be representative of real software engineering tasks that may arise in everyday programming. While coding challenges in interviews are meant to be solved within a few minutes and with limited code, they can still be related to solutions for real engineering issues. Here are some examples of how interview questions can be related to real engineering and development problems:

- You may be asked to write a program that sorts a list of objects by name or age. In a real-world situation, you may have an application that presents user data to an admin that must then be filtered and sorted in different ways.

- You may be asked to write a program that quickly determines what object in a list was created last and then removes it. This can be related to an application that presents events happening in real-time and needs to quickly change or remove the latest event.

- You may be asked to write a program that removes duplicate elements from a list. This can be related to a program that stores a birthday wishlist and prevents duplicate entries from being displayed if an original entry for an item already exists in the list.

As mentioned, coding challenges for bootcamp interviews are generally easier than coding challenges you may get in a job interview. For a bootcamp, knowledge of basic logic, simple looping, data structures such as arrays and objects, and an understanding of functional programming may be enough to pass. The challenges may be presented in a way that requires you to make use of different data structures and then combine them, along with several different lines of logical statements. This is to see if you have a basic grasp of programming, problem solving, and logical thinking.

---

[1] Google search for programmers and coding interviews: http://bit.ly/2dNSJ7j

In a job interview however, the coding challenges can be a bit harder and they may require use of more advanced data structures, complex algorithms, and clever insights to solve. We will cover some questions that can be asked in job interviews later on in the book, and you will see how they differ from coding challenges typically asked in coding bootcamp admissions. Understanding and being able to solve both types of coding challenges will prove to be incredibly useful in your coding education.

## 1.5.    What you'll learn in this book

From here on in, this book will provide you with some basic code examples and explanations. It will then list several different coding challenges that are a combination of the following topics:

- Fundamental data structures
- Working with hashtables
- Looping through arrays
- Functional programming

Every question will have an explanation detailing how to approach the problem and figure out a high-level solution, followed by the presentation of a working code solution. Some of the challenges will begin with a brute-force solution and then will slowly be modified towards a more efficient program. *All of the challenges will have working solutions in JavaScript, Python, and Ruby*.

My ultimate goal is not for you to memorize a bunch of solutions to common challenges. Instead, my goal is for you to learn *how* to approach coding challenges so that you can figure out what they are asking and what data structures and algorithms you need to solve them. Actually learning to code the solutions to problems takes time, practice, and a good grasp of the programming language you are using. In this book I hope to teach you common themes between most coding challenges and show you how specific data structures and algorithms can be used to solve almost every single challenge.

## 1.6.    Skill level required for these challenges

This book is intended for people who understand the basic syntax of JavaScript, Python, and/or Ruby, and have done some coding in one of those langauges. It will be easier to understand the challenge solutions if you have an understanding of the following topics:

- Loops
- Arrays
- String manipulation
- Objects and accessing different properties
- Hash tables
- Basic math functions (e.g. ceiling, floor, modulo)

If you still need to brush up on these topics in any of the three languages listed above, below are some resources you may find helpful:

**JavaScript**
1. http://eloquentjavascript.net (especially chapters 1, 3, 4 and 5)
2. https://coderbyte.com/course/learn-javascript-in-one-week (refresher in JavaScript basics)
3. https://www.codecademy.com/learn/javascript (if you're brand new to JavaScript)
4. https://teamtreehouse.com/library/javascript-basics

**Python**
1. https://learnpythonthehardway.org/book/
2. https://coderbyte.com/course/learn-python-in-one-week
3. https://www.codecademy.com/learn/python (if you're brand new to Python)
4. http://www.learnpython.org/ (small tutorials with code samples you can run online)

**Ruby**
1. http://tryruby.org/levels/1/ (fun interactive site to learn Ruby)
2. https://coderbyte.com/course/learn-ruby-in-one-week
3. https://www.codecademy.com/learn/ruby (if you're brand new to Ruby)
4. https://learnrubythehardway.org/book/

All of the challenges in Chapter 3 will be on a scale similar to the easy and medium challenges on Coderbyte (https://coderbyte.com/challenges). The only exception is the last challenge, or the "Three sum" problem. This one is a bit harder than the rest, but is a good challenge to study for a couple of reasons. It's an example of a challenge where a basic solution is easy to see at first (i.e., writing a few nested loops) but which can be improved by some clever manipulation of data structures and inputs to produce a much faster solution.

## 2.1.    Fundamental data structures you'll need for the bootcamp coding challenge

### Array

A data structure is a way of organizing data in a computer. A simple example of a data structure you may be familiar with already is the array. An array allows you to create and store a list of elements in some specific order for access later on. In some languages, the elements all have to be of the same type (e.g., string, integers, etc.), but in JavaScript, Python, and Ruby, you can have an array that stores elements of different types.

### Hash Table

The second most important data structure for coding challenges may very well be the hash table (also known as an 'associate array' or 'hash map'). It is a data structure that maps keys to values in an efficient way. Elements in hash tables are not necessarily stored in the order they are added; instead, they are accessed by their given keys. If you were checking for the existence of a certain item within an array, you would have to write a loop that checks every element in it and stops once the desired element was found. Even the built-in functions (e.g., **indexOf** in JavaScript, **in** in Python, **include?** in Ruby) that do this for us are written in a way that actually loops through the array. A hash table on the other hand does not need to loop to check if an element exists because the values are associated with given keys.[1] This means that a hash table lookup and insertion will run in constant time (more on this later).[2]

### Object

An object allows you to group together properties such as variables, arrays, and even other data structures and keep all of them in a sort of "container." In Python and Ruby, you first create a class which acts as a blueprint, and then you can instantiate objects based on the class. For example, you can create a Car class which will list all possible properties of a car, and then you can create several different Car objects. In JavaScript, this process is very similar except you do not create a class and then instantiate objects; rather, one object will act as the "root" (or prototype) object and then you can make copies of that object.

## 2.2.    What is Big-O?

One main aspect of computer science and programming is analyzing the speed at which our programs run given certain inputs. For example, imagine you wrote a program that reads in an array of names, sorts the array into alphabetical order, and then returns the result. If your program is given an array of 10 names, it most likely runs in less than half a second. But how long would it take for your program to run with 100,000 names, or even 1 million names? This is where the analysis of

---

[1] What is a HashTable Data Structure Youtube Video: http://bit.ly/2dZ5Q3E
[2] Stack Overflow posts about hash tables and running time: http://bit.ly/2dxFHFF and http://bit.ly/2dm1mD5

algorithms and Big-O comes into play. There are a ton of great resources out there that explain in rigorous detail how Big-O works both theoretically and practically, but I'll provide a short explanation so that later on you can have an understanding of the difference between a code solution that runs in $O(n^2)$ vs. $O(n)$.

Big-O is basically a tool that allows us to measure how algorithms respond to changes in input. In other words, how does the algorithm perform if the input $n$ is very small versus if $n$ is extremely large? The syntax for using Big-O is to use a variable (usually the letter $n$) which represents the length of the input, and place it within parentheses. Given that $O(n)$ means that the algorithms grows linearly with the input, you can imagine that we must perform a computational "step" for every element in the input. So if $n$ represents an array with 100 elements, then our algorithm will perform about 100 steps. If $n$ now grows to 100,000 elements, our algorithm will perform 100,000 steps.

Now imagine we have an algorithm that runs in $O(n^2)$. If $n$ represents an array with 100 elements, our algorithm will perform about 10,000 steps before finishing ($100^2$ = 10,000). If $n$ now grows to 100,000 elements, our algorithm will need to perform about 10,000,000,000 steps before finishing! Now you can see the extreme difference between $O(n)$ and $O(n^2)$ when $n$ is equal to 100,000 elements. Now let's say that each computational "step" takes about 1 millisecond to run (in reality, the time it takes to perform a simple computation is much faster than 1 millisecond on a modern computer). If the input $n$ is equal to an array with 100,000 elements, the $O(n)$ time algorithm will finishing running in about **100 seconds** (1 millisecond * 100,000), while the $O(n^2)$ algorithm will need about **115 days** before it finishes running. Given the option to choose an algorithm that runs in $O(n)$ or $O(n^2)$, we should absolutely always choose $O(n)$.

You can see a table of commonly used Big-O expressions here: http://bit.ly/2e2YlpC, and here is a good introductory article on the basics of Big-O: http://bit.ly/1PzbnX4.

PART 2
# CHALLENGES

---

*"One of my most productive days was throwing
away 1,000 lines of code."*

*- Ken Thompson*

## 3.1.    Fizz buzz

We will start off with the infamous Fizz buzz challenge. This challenge has been used frequently in the past as an initial coding challenge to filter out people who cannot write a simple solution. The general problem is:

*Print out all the numbers from 1 to 100. But for every number divisible by 3 print replace it with the word "Fizz," for any number divisible by 5 replace it with the word "Buzz" and for a number divisible by both 3 and 5 replace it with the word "FizzBuzz."*

So your program should output:

```
1
2
Fizz
4
Buzz
Fizz
7
.
.
.
```

Let us examine how we can solve this problem. The first thing we notice is that we will definitely need to write a loop because the problem asks us to print numbers within a range. Then we need to check every number we encounter whether it is divisible by 3, 5, or 3 and 5. This means we'll most likely need at least 3 conditional statements. To check if a number is divisible by some number X we will be using the modulo operator.[1] Then we need to either directly print the numbers, or we can store all of the numbers in an array and then return the final array. Below is a sample solution with some comments:

[1] Stack Overflow post explaining how to use the modulo operator: http://bit.ly/2dxq9Ss

*JavaScript*

```javascript
function fizzbuzz(n) {

  // we will store the resulting numbers within an array
  let result = [];

  // loop from 1 to n
  for (let i = 1; i <= n; i++) {

    let add = '';

    // check if there is a remainder when dividing by 3, if not
    // then we know this number is divisible by 3
    if (i % 3 === 0) { add += 'Fizz'; }

    // check if divisible by 5
    if (i % 5 === 0) { add += 'Buzz'; }

    // not divisible by either 3 or 5
    if (add === '') { result.push(i); }
    else { result.push(add); }

  }

  return result;

}
```

*Python*

```python
def fizzbuzz(n):

  # we will store the resulting numbers within an array
  result = []

  # loop from 1 to n
  for i in range(1, n + 1):

    add = ''

    # check if there is a remainder when dividing by 3, if not
    # then we know this number is divisible by 3
    if i % 3 == 0:
      add += 'Fizz'

    # check if divisible by 5
    if i % 5 == 0:
      add += 'Buzz'

    # not divisible by either 3 or 5
    if add == '':
      result.append(i)
    else:
      result.append(add)

  return result
```

*Ruby*

```ruby
def fizzbuzz(n)

  # we will store the resulting numbers within an array
  result = []

  # loop from 1 to n
  (1..n).each do |i|

    add = ''

    # check if there is a remainder when dividing by 3, if not
    # then we know this number is divisible by 3
    if i % 3 == 0
      add += 'Fizz'
    end

    # check if divisible by 5
    if i % 5 == 0
      add += 'Buzz'
    end

    # not divisible by either 3 or 5
    if add == ''
      result.push(i)
    else
      result.push(add)
    end

  end

  return result

end
```

## 3.2. Two sum problem

The two sum problem is a classic coding interview question asked at both coding bootcamps and job interviews. The problem is as follows:

*You are given an array and some number S. Determine if any two numbers within the array sum to S.*

We'll first cover the most intuitive way to solve this problem. That is to simply loop through the array and for each element check if there is a second element that when both are summed are equal to S. This will require us to write a nested loop.[1] This solution will run in $O(n^2)$ time though because of the nested loop. We can actually do better and write an algorithm that runs only in $O(n)$ time but uses some extra space.

---

[1] Post on "Why are nested loops considered bad practice?" http://bit.ly/2dL7w08

We will use a hash table where insertion and lookup run in constant $O(1)$ time. As we loop through the array we add each element into a hash table. Then we check to see if S minus the current elements exists in the hash table as we are looping.

*JavaScript*

```javascript
function twoSum(arr, S) {

  let hashTable = {};

  // check each element in array
  for (let i = 0; i < arr.length; i++) {

    // calculate S – current element
    let sumMinusElement = S – arr[i];

    // check if this number exists in hash table
    if (hashTable[sumMinusElement] !== undefined) {
      return true;
    }

    // add the current number to the hash table
    hashTable[arr[i]] = true;

  }

  return false;

}
```

*Python*

```python
def twoSum(arr, S):

  hashTable = {}

  # check each element in array
  for i in range(0, len(arr)):

    # calculate S minus current element
    sumMinusElement = S – arr[i]

    # check if this number exists in hash table
    if sumMinusElement in hashTable:
      return True

    # add the current number to the hash table
    hashTable[arr[i]] = True

  return False
```

*Ruby*

```ruby
def twoSum(arr, s)

  hashTable = Hash.new

  # check each element in array
  arr.each_with_index do |v, i|

    # calculate s minus current element
    sumMinusElement = s - arr[i]

    # check if this number exists in hash table
    if hashTable.key?(sumMinusElement)
      return true
    end

    # add the current number to the hash table
    hashTable[arr[i]] = true

  end

  return false

end
```

### 3.3.    Calculcate the sum of nested arrays

This problem asks you to sum up all of the numbers within an array, but the array may also contains other arrays with numbers. This is what we call a nested array. For example:

```
[1, 1, 1, [3, 4, [8]], [5]]
```

is a nested array and summing all the elements should produce 23. We will solve this problem by implementing a recursive function where if an array is encountered *within* an array, we simply sum all of the inner arrays elements and then add that to the final result. For example, with the code solution below applied to the above array, the following steps would take place:

1.  result = **1**
2.  result **= 1 + 1 = 2**
3.  result = **2 + 1 = 3**
4.  result = **3 + sumNested([3, 4, [8]])**
    4.1.    result = **3**
    4.2.    result = **3 + 4**
    4.3.    result = **7 + sumNested([8])**
        4.3.1.   result = **8**
5.  result = **3 + 15 = 18**
6.  result = **18 + sumNested([5])**
    6.1.    result = **5**
7.  result = **18 + 5 = 23**

The algorithm below simply loops through the entire array, and if a nested array is encounterd, it loops through all of its elements as well. The running time of this algorithm is therefore $O(n)$ where $n$ is the length of the entire list of elements.

*JavaScript*

```javascript
function sumNested(arr) {

  let result = 0;

  // sum up all the numbers in array
  for (let i = 0; i < arr.length; i++) {

    // if element is a nested array, sum all of its elements
    if (typeof arr[i] !== 'number') {
      result += sumNested(arr[i]);
    } else {
      result += arr[i];
    }

  }

  return result;

}
```

*Python*

```python
def sumNested(arr):

  result = 0

  # sum up all the numbers in array
  for i in range(0, len(arr)):

    # if element is a nested array, sum all of its elements
    if type(arr[i]) is not int:
      result += sumNested(arr[i])
    else:
      result += arr[i]

  return result
```

*Ruby*

```ruby
def sumNested(arr)

  result = 0

  # sum up all the numbers in array
  arr.each_with_index do |val, i|

    # if element is a nested array, sum all of its elements
    if !val.instance_of? Fixnum
      result += sumNested(val)
    else
      result += val
    end

  end

  return result

end
```

## 3.4.    Calculcate the angle on a clock

This problem asks you to determine the angle between the to hands on a clock. For example, if the minute hand is on the 3 and the hour hand is on the 12, then this forms a 90 degree angle. If the hour hand is slightly past the 2 and the minute hand is on the 4, then the angle formed between the hands is 50 degrees (image below for reference [Clock angle problem on Wikipedia: http://bit.ly/2dxFmmt]).
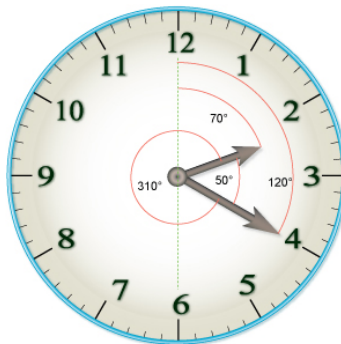


We will solve this problem by first calculating the angles for the hour hand and minute hand separately. The minute hand is the easiest and that is simply:

*Minute hand angle* = 6 * minutes

because there are 360 degrees total on the clock and there are a total of 60 minutes, so 360 / 60 = 6 degrees per minute. The hour hand is a bit trickier because it moves slightly every time the minute hand moves. Therefore, to calculate the angle of the hour hand we need to take into account how much the minute hand has moved.

First, we can perform the same calcuclation we did above but with different values. The hour hand moves around the entire clock after 12 hours, of 720 minutes, which gives us 360 / 720 = 0.5 per minute. We also need to account for how much the minute hand has moved. The full equation is below:

*Hour hand angle* = 0.5 * change in minutes = 0.5 * (60 * hour + minutes)

The angle between these two hands will therefore be the absolute value of the hour hand angle minus the minute hand angle. But one caveat, and that is if the number is greater than 180 degrees we actually need to subtract the value from 360 because the hands are on the opposite side of the clock.

*JavaScript*

```javascript
function clockAngle(hour, min) {

  var h = 0.5 * (60 * hour + min);
  var m = 6 * min;
  var angle = Math.abs(h - m);
  return (angle > 180) ? 360 - angle : angle;

}
```

*Python*

```python
def clockAngle(hour, mins):

  h = 0.5 * (60 * hour + mins)
  m = 6 * mins
  angle = abs(h - m)

  if angle > 180:
    return 360 - angle
  else:
    return angle
```

*Ruby*

```ruby
def clockAngle(hour, min)

  h = 0.5 * (60 * hour + min)
  m = 6 * min
  angle = (h - m).abs

  if angle > 180
    return 360 - angle
  else
    return angle
  end

end
```

### 3.5.    Determine if N is a prime number

This is a classic coding questins that asks you to write a program to determine whether or not some input N is a prime number. A prime number is a number that is divisible only by 1 and itself. The first few prime numbers are: 2, 3, 5, 7, 11, 17, …

The simplest way to solve this challenge is to first see that the only way for a number to be a prime is for it to have no divisors between 2 and itself. For example, to check if 9 is prime you would check if 9 is divisible by 2, if 9 is divisible by 3, etc. To check if a number is exactly divisible by some other number we are going to using the modulo operation to see if there is a remainder after the divison. For example, 9 modulo 2 gives us 1, but then 9 % 3 is 0 so we know that 9 is not a prime number because it is divisble by 3.

The solution for this problem would therefore be structured the following way: Write a loop from 2 to N to check if (N modulo x), where x is every number we are checking, is equal to 0. If so, then the number is prime, otherwise it is not. But there is a tiny detail we can implement to make the algorithm run a bit faster. The detail is that if N is not a prime number, then there are two numbers when multipled give N: a * b = N. The detail is that one of the numbers, a or b, will always be less than the square root of N.[1] With this detail in mind, we don't need to loop from 2 to N now, we can simply loop from 2 to the square root of N. The running time will therefore be $O(\sqrt{n})$.

*JavaScript*

```javascript
function isprime(n) {

  // all numbers less than 2 are not primes
  if (n < 2) { return false; }

  // loop from 2 to sqrt(n)
  for (let i = 2; i <= Math.ceil(Math.sqrt(n)); i++) {

    // check if (n mod i) is equal to 0, if so then there are
    // two numbers, a and b, that can multiply to give n
    if (n % i === 0) { return false; }

  }

  return true;

}
```

[1] Stack Overflow post and comment on "Calculcating and printing the Nth prime number" http://bit.ly/2e7gzLi

*Python*

```python
import math

def isprime(n):

  # all numbers less than 2 are not primes
  if n < 2:
    return False

  # loop from 2 to sqrt(n)
  for i in range(2, int(math.ceil(math.sqrt(n)))):

    # check if (n mod i) is equal to 0, if so then there are
    # two numbers, a and b, that can multiply to give n
    if n % i == 0:
      return False

  return True
```

*Ruby*

```ruby
def isprime(n)

  # all numbers less than 2 are not primes
  return false if n < 2

  # loop from 2 to sqrt(n)
  (2..Math.sqrt(n).round).each do |i|

    # check if (n mod i) is equal to 0, if so then there are
    # two numbers, a and b, that can multiply to give n
    return false if n % i == 0

  end

  return true

end
```

## 3.6.    Implement map and filter

Map and filter are common functional programming methods that you've most likely used when coding. They are both functions that take in a list, perform some operation on that list without changing the original list, and then return a new lists. The functions do not change any other variables and do not touch anything else except those lists they were given. JavaScript, Python, and Ruby all have their own built-in versions of these functions, but we are going to impement our own.

**Map** works by taking a list and a function, and it applies the function to each element in the list and returns a new list. For example, you may want to square every number in an array or append a string to every element in an array. We want an implementation where we can pass in two parameters, one being the array and the second being some function that will be *mapped* onto every element.

**Filter** works by taking a list and a conditional statement, and it returns a new list where every element in the original list passes the conditional (returns true). For example, you may have a list of ages and you want a new list of ages where each one is between 21 and 35. We want an implementation where, similar to the map function, we pass in a list and a function that contains within it a conditional statement.

Some extra resources on functional methods are listed below:

**JavaScript**
http://cryto.net/~joepie91/blog/2015/05/04/functional-programming-in-javascript-map-filter-reduce/

**Python**
http://blog.lerner.co.il/implementing-map-reduce/

**Ruby**
https://mauricio.github.io/2015/01/12/implementing-enumerable-in-ruby.html
http://augustl.com/blog/2008/procs_blocks_and_anonymous_functions/

*JavaScript*

```javascript
function map(arr, fn) {

  let result = [];

  // apply the function to each element and store the result
  for (let i of arr) {
    let applied = fn(i);
    result.push(applied);
  }

  return result;

}

// usage
let square = (x) => x * x;
let addZeros = (x) => parseInt(x += '00');

map([1, 2, 3, 4], square); // => [1, 4, 9, 16]
map([1, 2, 3, 4], addZeros); // => [100, 200, 300, 400]
```

```javascript
function filter(arr, fn) {

  let result = [];

  // pass the element to the function and check
  // if the result comes back true
  for (let i of arr) {
    let check = fn(i);
    if (check) { result.push(i); }
  }

  return result;

}

// usage
let isPositive = (x) => x > 0;
filter([-2, 4, 5, 8, -44, -6], isPositive); // => [4, 5, 8]
```

*Python*

```python
def map(arr, fn):

  result = []

  # apply the function to each element and store the result
  for i in arr:
    applied = fn(i)
    result.append(applied)

  return result

# usage
square = lambda x: x * x
addZeros = lambda x: int(str(x) + '00')

map([1, 2, 3, 4], square) # => [1, 4, 9, 16]
map([1, 2, 3, 4], addZeros) # => [100, 200, 300, 400]
```

```python
def filter(arr, fn):

  result = []

  # pass the element to the function and check
  # if the result comes back true
  for i in arr:
    check = fn(i)
    if check:
      result.append(i)

  return result

# usage
isPositive = lambda x: x > 0
filter([-2, 4, 5, 8, -44, -6], isPositive); # => [4, 5, 8]
```

*Ruby*

```ruby
def map(arr, fn)

  result = []

  # apply the function to each element and store the result
  arr.each do |i|
    applied = fn.call(i)
    result.push(applied)
  end

  return result

end

square = lambda { |x| x * x }
addZeros = lambda { |x| x.to_s + '00' }

p map([1, 2, 3, 4], square) # => [1, 4, 9, 16]
p map([1, 2, 3, 4], addZeros) # => [100, 200, 300, 400]
```

```ruby
def filter(arr, fn)

  result = []

  # pass the element to the function and check
  # if the result comes back true
  arr.each do |i|
    check = fn.call(i)
    if check
      result.push(i)
    end
  end

  return result

end

# usage
isPositive = lambda { |x| x > 0 }
p filter([-2, 4, 5, 8, -44, -6], isPositive); # => [4, 5, 8]
```

**3.7.** **Remove set of characters from a string**

These types of challenges are very common for people learning to code. The problem description is:

*You are given an array of characters and a string S. Write a function to return the string S with all the characters from the array removed.*

The first thing to notice is that we'll need to loop through the entire string S *and* loop through the array of characters because we'll need to find the characters to actually remove from the string. There are two ways to construct the loops:

1. Loop through the array of characters and for each chracter, find all of the occurences in S and remove them.

2. Loop through the string S and if the current character exists in the array of characters somewhere, remove it.

We're going to go with the second option because we can improve the algorithm by actually storing all of the characters in the array in a hash table. This will allow us to loop through the string S and determine if the current character needs to be removed by checking if it exists in the hash table. Remember from before, checking if an element exists in a hash table is done in constant time so the running time of our algorithm will be $O(n)$ where $n$ is the length of the string S plus some preprocessing where we loop through the array of characters and store them in a hash table.

*JavaScript*

```javascript
function removeChars(arr, string) {

  // store characters of arr in a hash table
  var hashTable = {};
  for (let c of arr) {
    hashTable[c] = true;
  }

  // loop through the string and check if the character exists in
  // the hash table, if so, do not add it to the result string
  let result = '';
  for (let c of string) {
    if (hashTable[c] === undefined) {
      result += c;
    }
  }

  return result;

}

// usage
removeChars(['h', 'e', 'w', 'o'], "hello world"); // => "ll rld"
```

*Python*

```python
def removeChars(arr, string):

  # store characters of arr in a hash table
  hashTable = {}
  for c in arr:
    hashTable[c] = True

  # loop through the string and check if the character exists in
  # the hash table, if so, do not add it to the result string
  result = ''
  for c in string:
    if c not in hashTable:
      result += c

  return result

# usage
print removeChars(['h', 'e', 'w', 'o'], 'hello world') # => "ll rld"
```

*Ruby*

```ruby
def removeChars(arr, string)

  # store characters of arr in a hash table
  hashTable = Hash.new
  arr.each { |c| hashTable[c] = true }

  # loop through the string and check if the character exists in
  # the hash table, if so, do not add it to the result string
  result = ''
  string.split('').each do |c|
    if !hashTable.key?(c)
      result += c
    end
  end

  return result

end

# usage
puts removeChars(['h', 'e', 'w', 'o'], 'hello world') # => "ll rld"
```

### 3.8.    Check if valid number of parenthesis

This problem asks you to determine if there is a valid number of matching parenthsis in a string, or more formally:

*You are given a string with the symbols ( and ) and you need to write a function that will determine if the parenthsis are correctly nested in the string which means every opening ( has a closing )*

There are countless ways to actually nest parenthsis and have them be valid, for example:

( )
( ( ) )
( ) ( ) ( )
( ( ( ) ( ) ) )

Below are examples of some invalid matchings:

( ( )
( ( ( (
( ) ) ( )
( ) ( ) ( ) )

The first thing to notice is that it doesn't matter how many parenthesis we actually have, what matters is that every single opening parenthsis "(" at some point has a closing parenthesis ")." What we can do to solve this challenge is to maintain a counter and every single time we encounter an opening symbol we add 1 to it. Then every time we encounter a closing symbol we remove 1. If all opening symbols have a matching symbol at some point in the string, the end result of the counter should be 0. If it is more or less we'll know that all parenthesis aren't properly matched. You can see that for the first 4 examples above, if we maintain a counter we will end up at 0 once we reach the end of the string.

*JavaScript*

```javascript
function matchingParens(string) {

  let counter = 0;

  for (let c of string) {
    if (c === '(') { counter += 1; }
    if (c === ')') { counter -= 1; }
  }

  return (counter === 0) ? true : false;

}
```

*Python*

```python
def matchingParens(string):

    counter = 0

    for c in string:
      if c == '(':
          counter += 1r
      elif c == ')':
          counter -= 1

    if counter == 0:
      return True
    else:
      return False
```

*Ruby*

```ruby
def matchingParens(string)

    counter = 0

    string.split('').each do |c|
      if c == '('
          counter += 1
      elsif c == ')'
          counter -= 1
      end
    end

    if counter == 0
      return true
    else
      return false
    end

end
```

### 3.9.    First non-repeating character

For this challenge you are given a string and you should return the first character that is unique in the entire string. For example:

- If string is "hello henry" then the first non-repeating character is the letter "o" because the first three characters in the string appear multiple times.

A simple solution to this challenge is to loop through the string, and for each character, loop through the rest of the string and check if that character appears somewhere else. Then return the first character that does not appear anywhere in the string except the current location. This solution will run in $O(n^2)$ though because of the nested loop, so there is definitely room for improvement.

We need to somehow store the letters and their frequencies in a data structure that'll allow us to check how many times a specific character occurred. To solve this challenge, we'll first loop through the string once and maintain a hash table that stores the count of each character. Then we'll loop through the string again and return the first character we encounter that has a count of 1 in the hash table, meaning it occurs only once in the string. This algorithm will run in $O(2n)$ because we are looping through the string twice, and this reduces to $O(n)$ because we drop the constant,[1] which is much faster than the nested loop algorithm from above.

*JavaScript*

```javascript
function firstNonrepChar(string) {

  let hashTable = {};

  // store each character in the hash table with
  // the frequency of times it occurs
  for (let c of string) {
    if (hashTable[c] === undefined) {
      hashTable[c] = 1;
    } else {
      hashTable[c] += 1;
    }
  }

  // loop through string and return the first character
  // with a count of 1 in the hash table
  for (let c of string) {
    if (hashTable[c] === 1) {
      return c;
    }
  }

  // return -1 if no unique character exists
  return -1;

}
```

[1] Stack Overflow post on "Why is the constant always dropped from Big-O analysis?" http://bit.ly/2dNSM2X

*Python*

```python
def firstNonrepChar(string):

  hashTable = {}

  # store each character in the hash table with
  # the frequency of times it occurs
  for c in string:
    if c not in hashTable:
      hashTable[c] = 1
    else:
      hashTable[c] += 1

  # loop through string and return the first character
  # with a count of 1 in the hash table
  for c in string:
    if hashTable[c] == 1:
      return c

  # return -1 if no unique character exists
  return -1
```

*Ruby*

```ruby
def firstNonrepChar(string)

  hashTable = {}

  # store each character in the hash table with
  # the frequency of times it occurs
  string.split('').each do |c|
    if !hashTable.key?(c)
      hashTable[c] = 1
    else
      hashTable[c] += 1
    end
  end

  # loop through string and return the first character
  # with a count of 1 in the hash table
  string.split('').each do |c|
    if hashTable[c] == 1
      return c
    end
  end

  # return -1 if no unique character exists
  return -1

end
```

### 3.10.  Count words that have at least 3 continuous vowels

This challenge will require us to take a string, separate it into words, and then loop through the words and count how many words have at least 3 vowels. This will require us to first convert a string into an array of words, then loop through that array and for each word loop through its characters and determine how many vowels exist in it and whether or not they are all adjacent to each other. There are two ways we can count the number or vowels within a word:

1.  Loop through the string and maintain a counter.
2.  Perform a regular expression search that counts the vowels.

We are going to go with the second method of using a regular expression (regex) to count the vowels in a word and determine whether they are continuous. I'll leave solving this challenge via the first method as an exercise for the reader. The solution below splits the string into an array of words and then loops through that array applying a regex search on each word. A regex search words by trying to find a search pattern within a string.[1] The search pattern we will be using is to search for a set of characters, the vowes in this case, and check whether at least 3 of them occur next to each other.

*JavaScript*

```javascript
function threeVowels(string) {

  // split string into array of words
  let arr = string.split(' ');
  let count = 0;

  // this is the pattern we will be searching for
  // more on regex patterns here: https://mzl.la/1bMbpXP
  const pattern = /[aeiou]{3,}/gi;

  // loop through array of words
  for (let word of arr) {
    if (word.match(pattern) !== null) {
      count += 1;
    }
  }

  return count;

}
```

---

[1] What is a Regular Expression? http://bit.ly/2dL7WUh

*Python*

```python
import re

def threeVowels(string):

  # split string into array of words
  arr = string.split(' ')
  count = 0

  # this is the pattern we will be searching for
  # more on regex patterns here: http://bit.ly/RGnLh4
  # loop through array of words
  for word in arr:
    if re.search(r'[aeiou]{3,}', word) != None:
      count += 1

  return count
```

*Ruby*

```ruby
def threeVowels(string)

  # split string into array of words
  arr = string.split(' ')
  count = 0

  # this is the pattern we will be searching for
  # more on regex patterns here: http://bit.ly/2dAvCZl
  pattern = /[aeiou]{3,}/

  # loop through array of words
  arr.each do |word|
    if pattern.match(word) != nil
      count += 1
    end
  end

  return count

end
```

### 3.11. Remove all adjacent matching characters

This challenge asks you to remove all matching adjacent pairs of letters from a string and return the modified string. For example, if the string is "aaagykkok" then your program would return "agyok" because "aa" and "kk" had been removed.

The first thing to notice is that we can create a simple loop that checks each character in the string and checks if it is equal to the character right after it, if so, it skips both characters. It does this for every character with a special case added to check if the end of the string has been reached (because at the last character, we cannot check if it is equal to the next character because there is no *next* character once it reaches the end of the string).

*JavaScript*

```javascript
function removePairs(string) {

  // new string that will be returned
  let result = '';

  // loop through entire string
  for (let i = 0; i < string.length; i++) {
    // if on last character
    if (i === string.length - 1) {
      result += string[i];
    }
    // characters are not equal then add to new string
    else if (string[i] !== string[i + 1]) {
      result += string[i];
    }
    // if adjacent characters are equal to each other
    // skip the next character entirely
    else {
      i += 1;
    }
  }

  return result;

}
```

*Python*

```python
def removePairs(string):

  # new string that will be returned
  result = ''
  i = 0

  # loop through entire string
  while i < len(string):
    # if on last character
    if i == len(string) - 1:
      result += string[i]
    # characters are not equal then add to new string
    elif string[i] != string[i + 1]:
      result += string[i]
    # if adjacent characters are equal to each other
    # skip the next character entirely
    else:
      i += 1
    i += 1

  return result
```

*Ruby*

```ruby
def removePairs(string)

  # new string that will be returned
  result = ''
  i = 0

  # loop through entire string
  while i < string.length do
    # if on last character
    if i == string.length - 1
      result += string[i]
    # characters are not equal then add to new string
    elsif string[i] != string[i + 1]
      result += string[i]
    # if adjacent characters are equal to each other
    # skip the next character entirely
    else
      i += 1
    end
    i += 1
  end

  return result

end
```

### 3.12.   Find the majority element (element that appears more than n/2 times)

Finding the majority element in an array involves finding an element that appears strictly more than n/2 times where n is the size of the array. For example, in the array [1, 4, 5, 5, 5, 5] the element 5 appears 4 times and n/2 = 6/2 = 3, so the element 5 is the majority element. If on the other hand the array was [1, 4, 4, 5, 5] then the element 5 is not the majority element. There actually is no majority element because no element appears more than n/2 = 5/2 = 2 times.

To solve this challenge we can create a nested loop and in the outer loop go through each element, and then in the inner loop check all the other elements in the array and count the occurences of the current element. This would make the code very compact but the algorithm would run in $O(n^2)$ time because of the nested loop checking each of the n elements, and then in the inner loop checking them again: $n * n = n^2$. There is a faster way to solve this problem and that is to use the Majority Vote Algorithm (http://bit.ly/1KVeJTk). It works the following way:

1. Start with a candidate element that is empty and a count that is set to 0.
2. For each element in the array we check it against the candidate element.
   - If the candidate *element is blank* or the count is *equal to 0*, set the current element to be the candidate element and set the count to 1.
   - If the current element equals the candidate element, increase the count by 1.
   - If the current element does not equal the candidate element, decrease the count by 1.

This algorithm will run in linear time, $O(n)$, because it only loops through the whole array a single time maintaining two variables as it passes through.

*JavaScript*

```javascript
function majorityElement(arr) {

  let candidate = null;
  var count = 0;

  // 1. if candidate is null or count = 0 set candidate to the current element
  // 2. if candidate matches current element we increase the count
  // 3. if candidate does not match current element, decrease the count by 1
  for (let i = 0; i < arr.length; i++) {
    if (candidate === null || count === 0) {
      candidate = arr[i];
      count = 1;
    } else if (arr[i] === candidate) {
      count += 1;
    } else {
      count -= 1;
    }
  }

  // once we have a majority element we check
  // to make sure it occurs more than n/2 times
  count = 0;
  for (let el of arr) {
    if (el === candidate) {
      count += 1;
    }
  }

  return (count > Math.floor(arr.length / 2)) ? candidate : null;

}
```

*Python*

```python
import math

def majorityElement(arr):

  candidate = None
  count = 0

  # 1. if candidate is None or count = 0 set candidate to the current element
  # 2. if candidate matches current element we increase the count
  # 3. if candidate does not match current element, decrease the count by 1
  for i in range(0, len(arr)):
    if candidate is None or count == 0:
      candidate = arr[i]
      count = 1
    elif arr[i] == candidate:
      count += 1
    else:
      count -= 1

  # once we have a majority element we check
  # to make sure it occurs more than n/2 times
  count = 0
  for el in arr:
    if el == candidate:
      count += 1

  if count > math.floor(len(arr) / 2):
    return candidate
  else:
    return None
```

*Ruby*

```ruby
def majorityElement(arr)

  candidate = nil
  count = 0

  # 1. if candidate is nil or count = 0 set candidate to the current element
  # 2. if candidate matches current element we increase the count
  # 3. if candidate does not match current element, decrease the count by 1
  arr.each do |i|
    if candidate == nil || count == 0
      candidate = i
      count = 1
    elsif i == candidate
      count += 1
    else
      count -= 1
    end
  end

  # once we have a majority element we check
  # to make sure it occurs more than n/2 times
  count = 0
  arr.each do |el|
    if el == candidate
      count += 1
    end
  end

  if count > (arr.length / 2).floor
    return candidate
  else
    return nil
  end

end
```

## 3.13.  Switching light bulbs problem

This problem has a lot of different variations, but the one we will cover here is the following: Imagine there are 100 light bulbs, labeled from 1 to 100, lined up all set to off initially. There are also 100 people each numbered 1 to 100 as well. Person 1 will go through all the light bulbs and flip the switch turning all of them on. Then person number 2 will go through all the light bulbs and flip the switch on each $2^{nd}$ element turning them off, namely: light bulbs #2, #4, #6, #8, etc. Then person 3 will go and do the same for the $3^{rd}$ ligh bulb, $6^{th}$, $9^{th}$, etc. Then questions are usually asked about the light bulbs, for example:

- How many light bulbs will be on after 100 people have gone through them?
- What is the status of the Nth light bulb ($34^{th}$, $62^{nd}$, etc.)? Is it on or off?
- How many people need to go through the line of light bulbs until exactly K light bulbs are set to on?

To answer any of these questions, we'll write a function that goes through all N light bulbs and performs each operation for every person labeled from 1 to N. What we'll do for this problem is setup an array of N elements all set to false, which will represent a light bulb in the off position. Then we will loop through every person from 1 to N, and within that loop create another loop that will multiply each person's number by a number each time to change the following light bulbs:

- For i = 1, change 1, 2, 3, 4, etc.
- For i = 2, change 2, 4, 6, 8, etc.
- For i = 3, change 3, 6, 9, 12, etc.

*JavaScript*

```javascript
function lightBulbs(N) {

  // create N lightbulbs and set them to off
  let lightbulbs = [];
  for (let i = 0; i < N; i++) {
    lightbulbs.push(false);
  }

  // each person i labeled from 1 to N sets each kth
  // lightbulb on or off where k = 2*i, 3*i, etc.
  for (let i = 1; i <= N; i++) {
    let w = 1;
    let k = w * i;
    while (k <= N) {
      lightbulbs[k − 1] = !lightbulbs[k − 1];
      w += 1;
      k = w * i;
    }
  }

  return lightbulbs;

}
```

*Python*

```python
def lightBulbs(N):

  # create N lightbulbs and set them to off
  lightbulbs = [False for i in range(0, N)]

  # each person i labeled from 1 to N sets each kth
  # lightbulb on or off where k = 2*i, 3*i, etc.
  for i in range(1, N + 1):
    w = 1
    k = w * i
    while k <= N:
      lightbulbs[k − 1] = not lightbulbs[k − 1]
      w += 1
      k = w * i

  return lightbulbs
```

*Ruby*

```ruby
def lightBulbs(n)

  # create N lightbulbs and set them to off
  lightbulbs = Array.new(n, false)

  # each person i labeled from 1 to N sets each kth
  # lightbulb on or off where k = 2*i, 3*i, etc.
  (1..n).each do |i|
    w = 1
    k = w * i
    while k <= n do
      lightbulbs[k - 1] = !lightbulbs[k - 1]
      w += 1
      k = w * i
    end
  end

  return lightbulbs

end
```

## 3.14.  List of integers that overlap in two ranges

For this problem you'll need to find all the numbers that overlap between two ranges, for example: between the two ranges of [5, 20] and [17, 21] the overlapping integers are [17, 18, 19, 20, 21]. One way to solve this problem would be to create a list for each range and store each number within the respective lists. So for the above example, you would create two lists:

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[17, 18, 19, 20, 21]
```

Then you would just loop through one of the lists, and for each element check if it exists somewhere in the second list using a built in array search function. This solution would work and require little code, but it is not an efficient way to solve this problem. One reason is because there isn't really any point in actually creating the lists and taking up space with them, and the second reason is the running time of this algorithm would be about $O(n^2)$ because this function would have a nested loop. We can solve this problem in a more efficient way.

We will simply loop from the start of range 1 to the end of range 1. Then we will check if each of the numbers within that range are greater than the beginning of range 2 and lesser than the end of range 2. This algorithm will therefore run in $O(n)$ time where $n$ represents the length of one of the ranges. This algorithm only performs a simple check within the loop, namely, whether some element is smaller and greater than some other number – no searching required which is what speeds up the algorithm.

*JavaScript*

```javascript
function overlapping(range1, range2) {

  let overlap = [];

  // check whether each number within range 1
  // is within the numbers in range 2
  for (let i = range1[0]; i <= range1[1]; i++) {
    if (i >= range2[0] && i <= range2[1]) {
      overlap.push(i);
    }
  }

  return overlap;

}
```

*Python*

```python
def overlapping(range1, range2):

  overlap = []

  # check whether each number within range 1
  # is within the numbers in range 2
  for i in range(range1[0], range1[1] + 1):
    if i >= range2[0] and i <= range2[1]:
      overlap.append(i)

  return overlap
```

*Ruby*

```ruby
def overlapping(range1, range2)

  overlap = []

  # check whether each number within range 1
  # is within the numbers in range 2
  (range1[0]..range1[1]).each do |i|
    if i >= range2[0] && i <= range2[1]
      overlap.push(i)
    end
  end

  return overlap

end
```

### 3.15. Return mean, median, and mode of an array

This is more of a simpler question that doesn't require too much complex code to solve. It simply requires you to do 3 things, calculcate the mean which is the average of all the numbers, the median which is the middle number when the array is sorted, and the mode which is the number that appears the most.

To calculcate the average we will simply add up all the numbers and divide by the length of the array. To calculcate the median we will first sort the array and then return the middle number. Finally, the mode will require some extra work, but to find the number that appears most often we will use a hash table to store the number of times each number occurs and we will return the one with the highest occurences.

*JavaScript*

```javascript
function meanMedianMode(arr) {

  // to calculcate the mean we add up all the numbers
  // and divide by the length
  let mean = arr.reduce((prev, cur) => prev + cur) / arr.length;

  // to calculcate the median we need to sort the array
  // and return the middle element
  arr = arr.sort();
  let median = arr[Math.floor(arr.length / 2)];

  // to get the mode we will store all the elements in a hash table
  // and keep a count and also always maintain the largest count
  let mode = undefined;
  let hashTable = {};
  for (let i of arr) {
    hashTable[i] === undefined ? hashTable[i] = 1 : hashTable[i] += 1;
    if (mode === undefined || hashTable[i] > mode) {
      mode = i;
    }
  }

  return { 'mean': mean, 'median': median, 'mode': mode };
```

*Python*

```python
def meanMedianMode(arr):

  # to calculcate the mean we add up all the numbers
  # and divide by the length
  mean = sum(arr) / float(len(arr))

  # to calculcate the median we need to sort the array
  # and return the middle element
  arr = sorted(arr)
  median = arr[int(len(arr) / 2)]

  # to get the mode we will store all the elements in a hash table
  # and keep a count and also always maintain the largest count
  mode = None
  hashTable = {}
  for i in arr:
    if i in hashTable:
      hashTable[i] += 1
    else:
      hashTable[i] = 1
    if mode is None or hashTable[i] > mode:
      mode = i

  return { 'mean': mean, 'median': median, 'mode': mode }
```

*Ruby*

```ruby
def meanMedianMode(arr)

  # to calculcate the mean we add up all the numbers
  # and divide by the length
  mean = arr.inject(:+) / arr.length.to_f

  # to calculcate the median we need to sort the array
  # and return the middle element
  arr = arr.sort
  median = arr[(arr.length / 2).floor]

  # to get the mode we will store all the elements in a hash table
  # and keep a count and also always maintain the largest count
  mode = nil
  hashTable = {}
  arr.each do |i|
    if hashTable.key?(i)
      hashTable[i] += 1
    else
      hashTable[i] = 1
    end
    if mode == nil or hashTable[i] > mode
      mode = i
    end
  end

  return { :mean => mean, :median => median, :mode => mode }

end
```

### 3.16. Encode consonants within a string

This challenge asks you to take a string composed of only lowercase letters and space characters, for example "hello world" and replace every consonant in the string with the *next consontant* in the alphabet. So in the above example, the output should be "jemmo xosmf" and you can see that we left every vowel in place and only changed the consonants. You should notice that the last letter changed was from *d to f* and not from *d to e* because *e* is a vowel.

The first thing that would be helpful for this challenge is some sort of listing of all the consonants which would allow us to easily change one letter to the other by just moving one place over in the array. Then to solve the challenge we would just loop through the string, and if a letter is not a vowel we would find that letter in the array of consonants, move one place over and replace the letter in the string with that new letter. This would be a solution to the challenge, but the running time involves both the length of the string $n$ and the length of the consonant array $m$ because we are looping through the string while also searching for a letter in the array each time. The running time still reduces to $O(n)$ because the consonant array stays constant meaning it will never get shorter or longer because there is a set amount of constants in the alphabet. So this solution is pretty good, but we can actually do a bit better.

To make this solution a bit faster we can actualy get rid of the consonant array altogether. This is because we can convert each letter as we go to its character code number, add 1 to it, and then convert this number back to a letter to get the next letter in the alphabet. Then we simply perform an extra check to make sure the new letter is not a vowel, but if so, simply add 1 more to its character code to get the *next* letter.

Note: You can actually perform a modulo operation at some point in the code and therefore there will be no need for the special "z" character conditional at the beginning of the loop. I'll leave this as an exercise for the reader.

*JavaScript*

```javascript
function encodeConsonants(string) {

  let result = '';

  // store an array of vowels for use later
  const vowels = ['a', 'e', 'i', 'o', 'u'];

  // loop through entire string
  for (let i of string) {

    // special case for z
    if (i === 'z') {
      result += 'b';
      break;
    }

    // if letter is not a vowel or a space
    else if (vowels.indexOf(i) === -1 && i !== ' ') {

      // convert each letter to its character code
      let newCode = i.charCodeAt(0) + 1;

      // perform check to make sure new letter is not a vowel by seeing if
      // the new letter exists in an array of vowels
      if (vowels.indexOf(String.fromCharCode(newCode)) !== -1) {
        newCode += 1;
      }

      // get new letter and add to new string
      result += String.fromCharCode(newCode);

    }

    // otherwise character is a vowel or a space
    else {
      result += i;
    }

  }

  return result;

}
```

*Python*

```python
def encodeConsonants(string):

  result = ''

  # store an array of vowels for use later
  vowels = ['a', 'e', 'i', 'o', 'u']

  # loop through entire string
  for i in string:

    # special case for z
    if i == 'z':
      result += 'b'
      break

    # if letter is not a vowel or a space
    elif i not in vowels and i != ' ':

      # convert each letter to its character code
      newCode = ord(i) + 1

      # perform check to make sure new letter is not a vowel by seeing if
      # the new letter exists in an array of vowels
      if chr(newCode) in vowels:
        newCode += 1

      # get new letter and add to new string
      result += chr(newCode)

    # otherwise character is a vowel or a space
    else:
      result += i

  return result
```

*Ruby*

```ruby
def encodeConsonants(string)

  result = ''

  # store an array of vowels for use later
  vowels = ['a', 'e', 'i', 'o', 'u']

  # loop through entire string
  string.chars.each do |i|

    # special case for z
    if i == 'z'
      result += 'b'
      break

    # if letter is not a vowel or a space
    elsif vowels.include?(i) == false and i != ' '

      # convert each letter to its character code
      newCode = i.ord + 1

      # perform check to make sure new letter is not a vowel by seeing if
      # the new letter exists in an array of vowels
      if vowels.include?(newCode.chr)
        newCode += 1
      end

      # get new letter and add to new string
      result += newCode.chr

    # otherwise character is a vowel or a space
    else
      result += i
    end

  end

  return result

end
```

### 3.17.    Convert an array of strings into an object

This challenge doesn't strictly have a single output like the previous challenges, rather this challenge focuses on you using certain data structures correctly. Imagine you have several users entering information through a form, and on the back-end you get the information as a comma separated string of information. The information the user will enter in the form is: *Name*, *Email*, *Age*, and *Occupation* all in that order. Each user's piece of information will be separated by a comma, and each user will be separated by a space, but some pieces of information can be blank for a user (excluding the name), for example:

`"Daniel,me@test.com,56,Coder John,,,Teacher Michael,mike@test.com,,"`

You can see above that all the information exists for Daniel, but email and age are missing for John, and age and occupation are missing for Michael. You should write a function that will take in this string of user information, and create an object (key-value mapping) where the key is the person's name and the value will also be an object that stores the information of each user: email, age, and occupation. You can assume people's names will be unique.

To solve this challenge, we'll split the string (at the space character) into an array of different people, and then for each person we'll split the information (at the comma character). Then we will create a new user object and throw it into the global object that will store all the users.

*JavaScript*

```javascript
function convert(string) {

  // create empty object
  let obj = {};

  // split the string at each person
  const people = string.split(' ');

  // loop through all people
  for (let p of people) {

    // split information for each person
    const info = p.split(',');

    // store this information in the user object
    let userObj = {
      'email': info[1] || null,
      'age': parseInt(info[2]) || null,
      'occupation': info[3] || null
    };

    // store key-value in object of users now
    obj[info[0]] = userObj;

  }

  return obj;

}

// usage
let s = "Daniel,me@test.com,56,Coder John,,,Teacher Michael,mike@test.com,,";
let people = convert(s);

// testing
people['Daniel']['age']; // => 56
people['Michael']['occupation'] // => null
```

*Python*

```python
def convert(string):

  # create empty object
  obj = {}

  # split the string at each person
  people = string.split(' ')

  # loop through all people
  for p in people:

    # split information for each person
    info = p.split(',')

    # store this information in the user object
    userObj = {
      'email': info[1] or None,
      'age': int(info[2]) if info[2] else None,
      'occupation': info[3] or None
    }

    # store key-value in object of users now
    obj[info[0]] = userObj

  return obj

# usage
s = "Daniel,me@test.com,56,Coder John,,,Teacher Michael,mike@test.com,,"
people = convert(s)

# testing
people['Daniel']['age'] # => 56
people['Michael']['occupation'] # => None
```

*Ruby*

```ruby
def convert(string)

  # create empty object
  obj = {}

  # split the string at each person
  people = string.split(' ')

  # loop through all people
  people.each do |p|

    # split information for each person
    info = p.split(',')

    # store this information in the user object
    userObj = {
      'email' => info[1] || nil,
      'age' => info[2].to_i || nil,
      'occupation' => info[3] || nil
    }

    # store key-value in object of users now
    obj[info[0]] = userObj

  end

  return obj

end

# usage
s = "Daniel,me@test.com,56,Coder John,,,Teacher Michael,mike@test.com,,"
people = convert(s)

# testing
p people['Daniel']['age'] # => 56
p people['Michael']['occupation'] # => None
```

### 3.18.   Three sum problem

The three sum problem is very similar to the two sum problem we covered at the beginning of the chapter, except the problem statement now naturally changes to 3 elements instead of two:

*You are given an array and some number S. Determine if any three numbers within the array sum to S.*

As with the nested loop solution to the two sum problem, we can create a set of three nested loops that checks if any three elements in the array sum to S. The running time of this algorithm is $O(n^3)$ because of the nested loops. We can solve this problem more efficiently though with some modifications.

First we sort the array into ascending order. We can use a built-in sort function which will run in $O(n\log n)$.[1] Then we loop through each element in the array and for each of the elements we maintain two pointers (or indices), one from the (*current position + 1)* and the second pointer starting from the *end of the array*. Then we check to see if these 3 elements sum to S. One of 3 things will be true:

1.  If the current element plus the 2 other elements is greater than S, we decrease the pointer at the end of the array at 1.

2.  If the current element plus the other 2 elements is less than S, we increase the pointer at the beginning by 1.

3.  If the pointers end up meeting, then we know we cannot add 2 elements plus the current element to sum to S. We move on to the next element in the array and reset the pointers.

This algorithm needs to loop through every single element in the array, and for each element in the worst case we will loop through all the other elements until the pointers end up at the same point. This algorithm therefore runs in $O(n\log n)$ + $O(n^2)$ which reduces to $O(n^2)$.

---

[1] The built-in sorting functions in JavaScript, Python, and Ruby use either mergesort, quicksort, or some combination of them to create a fast sorting algorithm that runs in $O(n\log n)$ time.

*JavaScript*

```javascript
function threeSum(arr, S) {

  // sort the array
  var arr = arr.sort();

  // loop and check each element
  for (let i = 0; i < arr.length - 2; i++) {

    // start two pointers, one from the current position + 1
    // and the other at the end of the array
    let ptr_start = i + 1;
    let ptr_end = arr.length - 1;

    // check all other elements
    while (ptr_start < ptr_end) {

      let add = arr[i] + arr[ptr_start] + arr[ptr_end];

      // if we find a sum
      if (add === S) { return true; }

      // if the sum is < S
      else if (add < S) { ptr_start += 1; }

      // otherwise the sum is > S
      else { ptr_end -= 1; }

    }

  }

  return false;

}
```

*Python*

```python
def threeSum(arr, S):

  arr = sorted(arr)

  for i in range(0, len(arr) - 2):
    # start two pointers, one from the current position + 1
    # and the other at the end of the array
    ptr_start, ptr_end = i + 1, len(arr) - 1

    while ptr_start < ptr_end:
      add = arr[i] + arr[ptr_start] + arr[ptr_end]

      # if we find a sum
      if add == S:
        return True
      # if the sum < S
      elif add < S:
        ptr_start += 1
      # if the sum > S
      else:
        ptr_end -= 1

  return False
```

*Ruby*

```ruby
def threeSum(arr, S)

  arr = arr.sort()

  (0..arr.length - 2).each do |i|

    # start two pointers, one from the current position + 1
    # and the other at the end of the array
    ptr_start = i + 1
    ptr_end = arr.length - 1

    while ptr_start < ptr_end
      add = arr[i] + arr[ptr_start] + arr[ptr_end]

      # if we find a sum
      if add == S
        return true

      # if the sum < S
      elsif add < S
        ptr_start += 1

      # if the sum > S
      else
        ptr_end -= 1

      end

    end

  end

  return false

end
```

PART 3
# GENERAL INTERVIEW TIPS

*"Computer science is no more about computers
than astronomy is about telescopes."*

*- Edsger Dijkstra*

Staying calm in a coding interview and also being able to solve the challenge within a time limit is a skill that can be mastered with enough practice and some general tips. In the bulk of this book we've discussed over a dozen common coding questions asked in a coding bootcamp interview, and we also covered some background on coding bootcamps and algorithms. Below we'll provide some extra resources where you can learn about how to stay calm during a stressful coding question, what questions to ask the interviewer, how to answer certain questions effectively, etc.

## 4.1.    Personal help

For personal help, Coderbyte offers the Personalized Study Plan (https://coderbyte.com/study-plan) that will help create a personal step-by-step guide for you to help prepare for a coding interview to a bootcamp or job. It contains article, resources, challenges, and videos tailored to your goals and skill level and it's the best way to prepare for an upcoming coding interview at a top company or elite coding bootcamp such as Hack Reactor, Fullstack Acadaemy, Flatiron School, Grace Hopper, Codesmith, etc.

We provide access to an online community with our own Coderbyte Slack channel. As part of the Study Plan mentioned above, you will get access to the channel and you'll be able to ask questions and chat with fellow coders preparing for upcoming interviews.

Aside from these features, one of the best ways to prepare for any type of coding interview is to simply practice solving coding challenges, practice, and then practice some more. Coderbyte offers more than 160 coding challenges (https://coderbyte.com/challenges) that you can solve directly online. Then you can look at thousands of user solutions to see how others solved the same challenges.

## 4.2.    Interview preparation articles

- Technical Interview Questions, Prep and More by Hack Reactor: http://bit.ly/2e9niPs
- 10 Must-Asks in a Bootcamp Interview by Course Report: http://bit.ly/2etZKcp
- How to Ace the Web Developer Job Interview by Coding Dojo: http://bit.ly/21y4UBd
- 5 Tips to Prepare You for a Coding Bootcamp by Galvanize: http://bit.ly/2dNTuxi
- From Zero to Hack Reactor in 5 months blog post by Jennie Eldon: http://bit.ly/2dMjhT7
- What I Studied Before Applying to Hack Reactor post by Amira Anuar: http://bit.ly/2d8MJ7F
- Inside the Mind of Hackbright's Director of Admissions by Hackbright: http://bit.ly/2dNTwp4
- Compare top coding bootcamps tool by Thinkful: http://bit.ly/2dxFhiI