**Палиндром**  
  
Палиндром — слово, предложение или последовательность символов, которая абсолютно одинаково читается как в привычном направлении, так и в обратном. К примеру, “Anna” — это палиндром, а “table” и “John” — нет.  
  
***Постановка***  
  
Дана строка; нужно написать функцию, которая позволяет вернуть значение true, если строка является палиндромом, и false — если нет. При этом нужно учитывать пробелы и знаки препинания.  
  
*palindrome('racecar') === true  
palindrome('table') === false*  
  
***Разбираем задание***  
  
Основная идея здесь — перевернуть строку в обратном направлении. Если «реверсная» строка полностью идентична исходной, значит, мы получили палиндром и функция должна вернуть значение true. Если же нет — false.  
  
***Решение***  
  
Вот код, который позволяет решить палиндром.

const palindrome = str => {

*// turn the string to lowercase*

str = str.toLowerCase()

*// reverse input string and return the result of the*

*// comparisong*

return str === str.split('').reverse().join('')

}

Первый шаг — преобразование символов входной строки в нижний регистр. Это гарантия того, что программа будет сравнивать именно сами символы, а не регистр или еще что-то.  
  
Второй шаг — реверс строки. Это сделать несложно: необходимо преобразовать ее в массив посредством метода .split() (библиотека String). Потом мы переворачиваем массив, используя .reverse() (библиотека Array). Последний этап — преобразование обратного массива в строку при помощи .join() (библиотека Array).  
  
Теперь все, что нужно, — сравнить «обратную» строку с исходной, вернув результат true или false.  
  
**FizzBuzz**  
  
Одна из самых популярных на собеседованиях задач.  
  
***Постановка***  
  
Требуется написать функцию, выводящую в консоль числа от 1 до n, где n — это целое число, которая функция принимает в качестве параметра, с такими условиями:

* вывод fizz вместо чисел, кратных 3;
* вывод buzz вместо чисел, кратных 5;
* вывод fizzbuzz вместо чисел, кратных как 3, так и 5.

***Пример***  
  
*Fizzbuzz(5)*  
  
***Результат***  
  
*// 1  
// 2  
// fizz  
// 4  
// buzz*  
  
***Разбираем задание***  
  
Главное здесь — способ поиска кратных чисел с использованием JavaScript. Его можно реализовать при помощи оператора модуля или же остатка — %, который позволяет показать остаток при делении двух чисел. Если остаток 0, это означает, что первое число кратно второму.  
  
*12 % 5 // 2 -> 12 is not a multiple of 5  
12 % 3 // 0 -> 12 is multiple of 3*  
  
Так, если разделить 12 на 5, получаем 2 с остатком 2. Если же разделить 12 на 3, то получаем 4 с остатком 0. В первом случае 12 не кратно 5, во втором — 12 кратно 3.  
  
***Решение***  
  
Оптимальным решением будет следующий код:

const fizzBuzz = num => {

for(let i = 1; i <= num; i++) {

*// check if the number is a multiple of 3 and 5*

if(i % 3 === 0 && i % 5 === 0) {

console.log('fizzbuzz')

} *// check if the number is a multiple of 3*

else if(i % 3 === 0) {

console.log('fizz')

} *// check if the number is a multiple of 5*

else if(i % 5 === 0) {

console.log('buzz')

} else {

console.log(i)

}

}

}

Функция выполняет нужные проверки с использованием условных операторов и выдает результат, необходимый пользователю. В задаче стоит уделить внимание порядку операторов if…else: начинать с двойного условия (&&) и заканчивать случаем, когда кратные числа найти не получилось. В результате мы охватываем все варианты.  
  
**Анаграмма**  
  
Так называют слово, которое содержит все буквы другого слова в том же количестве, но ином порядке.  
  
***Постановка***  
  
Нужно написать функцию, которая проверяет, являются ли две строки анаграммами, причем регистр букв не имеет значения. Учитываются лишь символы; пробелы или знаки препинания в расчет не берутся.  
  
*anagram('finder', 'Friend') --> true  
anagram('hello', 'bye') --> false*  
  
***Разбираем задание***  
  
Здесь важно учитывать, что необходимо проверять каждую букву в двух входных строках и их количество в каждой строке.  
  
*finder --> f: 1 friend --> f: 1  
i: 1 r: 1  
n: 1 i: 1  
d: 1 e: 1  
e: 1 n: 1  
r: 1 d: 1*  
  
Для хранения данных анаграммы стоит выбрать такую структуру, как объектный литерал JavaScript. Ключ в этом случае — символ буквы, значение — количество ее повторений в текущей строке.  
  
Есть и другие условия:

* Нужно убедиться в том, что регистр букв при сравнении не учитывается. Просто преобразуем обе строки в нижний или верхний регистр.
* Исключаем из сравнения все не-символы. Лучше всего работать с [регулярными выражениями](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp).

***Решение***

*// helper function that builds the*

*// object to store the data*

const buildCharObject = str => {

const charObj = {}

for(let char of str.replace(/[^\w]/g).toLowerCase()) {

*// if the object has already a key value pair*

*// equal to the value being looped over,*

*// increase the value by 1, otherwise add*

*// the letter being looped over as key and 1 as its value*

charObj[char] = charObj[char] + 1 || 1

}

return charObj

}

*// main function*

const anagram = (strA, strB) => {

*// build the object that holds strA data*

const aCharObject = buildCharObject(strA)

*// build the object that holds strB data*

const bCharObject = buildCharObject(strB)

*// compare number of keys in the two objects*

*// (anagrams must have the same number of letters)*

if(Object.keys(aCharObject).length !== Object.keys(bCharObject).length) {

return false

}

*// if both objects have the same number of keys*

*// we can be sure that at least both strings*

*// have the same number of characters*

*// now we can compare the two objects to see if both*

*// have the same letters in the same amount*

for(let char in aCharObject) {

if(aCharObject[char] !== bCharObject[char]) {

return false

}

}

*// if both the above checks succeed,*

*// you have an anagram: return true*

return true

}

Обратите внимание на использование [Object.keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys) в сниппете выше. Этот метод возвращает массив, содержащий имена или ключи в таком же порядке, в каком они встречаются в объекте. В этом случае массив будет таким:  
  
*['f', 'i', 'n', 'd', 'e', 'r']*  
  
Таким образом, мы получаем свойства объекта без необходимости выполнять объемный цикл. В задаче можно использовать этот способ со свойством .length — для проверки того, есть ли в обеих строках одинаковое количество символов — это важная особенность анаграмм.  
  
***Поиск гласных***  
  
Достаточно простая задача, которая часто попадается на собеседованиях.  
  
***Постановка***  
  
Нужно написать функцию, принимающую строку в качестве аргумента и возвращающую количество гласных, которые содержатся в строке.  
Гласными являются «a», «e», «i», «o», «u».  
  
***Пример:***  
  
*findVowels('hello') // --> 2  
findVowels('why') // --> 0*  
  
***Решение***  
  
Вот самый простой вариант:

const findVowels = str => {

let count = 0

const vowels = ['a', 'e', 'i', 'o', 'u']

for(let char of str.toLowerCase()) {

if(vowels.includes(char)) {

count++

}

}

return count

}

Важно обратить внимание на использование метода .includes(). Он доступен и для строк, и для массивов. Его стоит применять для того, чтобы выявить, содержит ли массив определенное значение. Этот метод возвращает true, если массив содержит указанное значение, и false, если нет.  
  
Есть и более краткое решение проблемы:

const findVowels = str => {

const matched = str.match(/[aeiou]/gi)

return matched ? matches.length : 0

}

Здесь задействуется метод .match(), который позволяет реализовать эффективный поиск. Если регулярное выражение как аргумент метода обнаружено внутри указанной строки, то возвращаемым значением становится массив совпадающих символов. Ну а если совпадений нет, то .match() возвращает null.  
  
**Фибоначчи**  
  
Классическая задача, которую можно встретить на собеседованиях самого разного уровня. Стоит напомнить, что последовательность Фибоначчи — это ряд чисел, где каждое последующее является суммой двух предыдущих. Так, первые десять чисел выглядят следующим образом: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.  
  
***Постановка***  
  
Нужно написать функцию, которая возвращает n-ную запись в определенной последовательности, причем n — число, которое передается в качестве аргумента функции.  
  
*fibonacci(3) // --> 2*  
  
Эта задача включает прохождение по циклу такое количество раз, которое указано в аргументе, с возвратом значения на соответствующей позиции. Этот способ постановки задачи требует использования циклов. Если вместо этого вы используете рекурсию, это может понравиться интервьюеру и дать вам несколько дополнительных очков.  
  
***Решение***

const fibonacci = num => {

*// store the Fibonacci sequence you're going*

*// to generate inside an array and*

*// initialize the array with the first two*

*// numbers of the sequence*

const result = [0, 1]

for(let i = 2; i <= num; i++) {

*// push the sum of the two numbers*

*// preceding the position of i in the result array*

*// at the end of the result array*

const prevNum1 = result[i - 1]

const prevNum2 = result[i - 2]

result.push(prevNum1 + prevNum2)

}

*// return the last value in the result array*

return result[num]

}

В массиве результатов первые два числа содержатся в ряду, поскольку каждая запись в последовательности состоит из суммы двух предыдущих чисел. В самом начале двух чисел, которые можно взять для получения следующего числа нет, поэтому цикл не может сгенерировать их в автоматическом режиме. Но, как мы знаем, первые два числа — всегда 0 и 1. Поэтому инициализировать массив результатов можно вручную.  
  
Что касается рекурсии, то здесь все проще и сложнее одновременно:

const fibonacci = num => {

*// if num is either 0 or 1 return num*

if(num < 2) {

return num

}

*// recursion here*

return fibonacci(num - 1) + fibonacci(num - 2)

}

Мы продолжаем вызывать fibonacci(), передавая все меньшие числа в качестве аргументов. Останавливаемся в случае, когда переданный аргумент равен 0 или 1.

Вывод

Скорее всего, вы уже сталкивались с какой-либо из указанных задач, если проходили собеседования на работу frontend- или JavaScript-разработчика (особенно если это уровень junior). Но если они вам и не попадались, то могут пригодиться в будущем — как минимум для общего развития.