**Выров Максим Пдо32**

**Основы JS**

**Задание 1**

Задание: Нужно написать функцию, которая возвращает тип входного аргумента

На входе: один аргумент, любого значения

На выходе: строка, с типом входного аргумента

Код:

**function getArgumentType(arg) {**

**return typeof arg;**

**}**

**console.log(getArgumentType(5)); // "number"**

**console.log(getArgumentType("Hello")); // "string"**

**console.log(getArgumentType(true)); // "boolean"**

**console.log(getArgumentType(null)); // "object"**

**console.log(getArgumentType(undefined)); // "undefined"**

**Вывод:**



**Задание 2**

Задание: Что выведется в консоль и почему?

**const a = () => console.log("a");**

**const b = () => console.log("b");**

**const c = a || b ? a() : b();**

**console.log(c);**

**Вывод:**



**Пояснение:**

1. Сначала создаются две стрелочные функции a и b, которые выводят в консоль строки "a" и "b" соответственно.

2. Затем происходит вычисление выражения a || b. Оператор || выполняет логическое ИЛИ. В данном случае, так как функции a и b являются истинными значениями, результатом выражения будет a, так как a находится слева от оператора.

3. Затем вызывается функция a() и выводится в консоль "a".

4. Однако, так как функция a() не возвращает явно никакого значения, результатом вызова будет undefined.

5. Итоговый результат, который присваивается переменной c, будет undefined.

6. Наконец, выводится в консоль значение переменной c, которое будет "undefined".

**Функции**

**Задание 1**

Задание: Нужно реализовать функцию, возвращающая сумму всех аргументов,

если тип аргумента - число. Вызовов ВСЕГДА 5

Код:

**function count(num) {**

**if (typeof num !== 'number') {**

**return count;**

**}**

**let sum = num;**

**function nextCount(nextNum) {**

**if (typeof nextNum === 'number') {**

**sum += nextNum;**

**}**

**return nextCount;**

**}**

**nextCount.toString = function () {**

**return sum.toString();**

**};**

**return nextCount;**

**}**

**console.log(count(3)(2)(1)(5)(7).toString());**

**console.log(count(3)(null)(1)('asad')().toString());**

**Вывод:**



**Объекты и Массивы**

**Задание 1**

Задание: Реализовать функцию convertArrToObj, которая на вход принимает

массив значений, а на выход отдает объект, где ключ - это индекс элемента в

массиве, а значение - сам элемент

Код:

**function convertArrToObj(arr) {**

**const obj = {};**

**for (let i = 0; i < arr.length; i++) {**

**obj[i] = arr[i];**

**}**

**return obj;**

**}**

**const arr = [1, null, 'test', undefined];**

**const obj = convertArrToObj(arr);**

**console.log(obj); // { 0: 1, 1: null, 2: 'test', 3: undefined }**

**Вывод:**



**Задание 2**

Задание: Требуется реализовать функцию countFromArr, которая позволяет

подсчитать значения массива. На вход поступает массив, на выход получаем

объект, где ключ - это элемент массива, а значение - его кол-во в массиве

Код:

**function countFromArr(arr) {**

**const countObj = {};**

**for (let i = 0; i < arr.length; i++) {**

**const element = arr[i];**

**if (countObj[element]) {**

**countObj[element]++;**

**} else {**

**countObj[element] = 1;**

**}**

**}**

**return countObj;**

**}**

**const arr = [1, 1, 1, 'test', 'test'];**

**const res = countFromArr(arr);**

**console.log(res); // { 1: 3, test: 2 }**

**Вывод:**



**Задание 3**

Задание: Требуется реализовать функцию groupByField, которая позволяет

сгруппировать массив объектов по определенному полю этого объекта.

На вход поступает массив объектов и строка с названием нужного поля, на выход

объект - где ключом является значение поля объекта, а значением - массив

объектов где значение в поле равно этому ключу

Код:

**function groupByField(arr, field) {**

**const result = {};**

**for (const obj of arr) {**

**const value = obj[field];**

**if (value) {**

**if (result[value]) {**

**result[value].push(obj);**

**} else {**

**result[value] = [obj];**

**}**

**}**

**}**

**return result;**

**}**

**const arr = [{test: 1}, {test: 2}, {test: 3}, {test: 1}, {test: 1}];**

**const res = groupByField(arr, 'test');**

**console.log(res); // { 1: [{test: 1}, {test: 1}, {test: 1}], 2: [{test: 2}], 3: [{test: 3}] }**

**Вывод:**



**Прототипы, наследование**

**Задание 1**

Задание: Реализовать функцию plus(

Код:

**Number.prototype.plus = function (num) {**

**return this + num;**

**};**

**console.log((1).plus(2));**

**Вывод:**



**Асинхронная работа в JS"**

**Задание 1**

Задание: Что выведется в консоль?

**console.log("start");**

**setTimeout(() => console.log("timeout"), 0);**

**new Promise((resolve, reject) => {**

**console.log("promise constructor");**

**reject();**

**})**

**.then(() => console.log("p1"))**

**.catch(() => console.log("p2"))**

**.catch(() => console.log("p3"))**

**.then(() => console.log("p4"))**

**.then(() => console.log("p5"));**

**console.log("final");**

**Вывод:**



**Объяснение:**

1. Строка "start" будет выведена первой, так как является синхронным кодом.

2. Затем будет выполнено создание нового промиса new Promise((resolve, reject) => { ... }). Функция-исполнитель внутри промиса синхронно выполняется, поэтому строка "promise constructor" будет выведена следующей.

3. Далее вызывается метод reject(), который вызывает отклонение промиса.

4. Затем идет вызов метода then() на промисе, но поскольку промис был отклонен, переходим к следующему доступному блоку catch().

5. Блок catch() перехватывает отклонение промиса и выводит строку "p2".

6. После блока catch() следует блок then(), который не был пропущен благодаря обработке исключений. Блок then() выводит строку "p4".

7. Затем идет блок then(), который выводит строку "p5".

8. Следующая строка "timeout" будет выведена, потому что она относится к отложенному выполнению, ожидающему выполнения в очереди событий.

9. В конце выводится строка "final", так как это синхронный код, следующий после цепочки промисов.

Важно отметить, что момент выполнения колбэка таймера setTimeout() с нулевой задержкой может немного отличаться в различных средах выполнения JavaScript, поэтому порядок вывода "timeout" и "final" может незначительно меняться