**Шилдаев Алексей ПДо-32**

**Основы JS Задание 1**

Функция, которая возвращает тип входного аргумента

function getArgumentType(arg) {

return typeof arg;

}

console.log(getArgumentType(5));

console.log(getArgumentType("Hello"));

console.log(getArgumentType(true));

console.log(getArgumentType(null));

console.log(getArgumentType(undefined));

Вывод

node /tmp/45ml5kUUgx.js

number

string

boolean

object

undefined

**Задание 2**

*В консоль будет выведено:*

*a*

*undefined*

При выполнении строки `const c = a || b ? a() : b();` происходит следующее:

1. `a` и `b` - это функции, которые выводят в консоль сообщения "a" и "b" соответственно.

2. Выражение `a || b` оценивается как функция `a`, так как в логическом операторе `||` вернется первый истинный операнд, и в данном случае `a` интерпретируется как истинное значение.

3. Выражение `a()` вызывает функцию `a` и выводит в консоль сообщение "a".

4. Значение, которое возвращает вызов функции `a()`, не указано явно, поэтому по умолчанию будет возвращено `undefined`.

5. Результат операции зависит от значения, возвращаемого вызовом функции `a()`, поскольку это `undefined`, то выполнится вторая часть тернарного оператора `: b()`.

6. Вызывается функция `b` и выводится в консоль сообщение "b".

7. Значение, которое возвращает вызов функции `b()`, также не указано явно, и поэтому по умолчанию возвращается `undefined`.

8. Переменная `c` получает значение `undefined`.

9. В консоль выводится значение `undefined`.

**Функции Задание 1**

```javascript

function count(number) {

let sum = number || 0;

function innerCount(nextArgument) {

if (typeof nextArgument === 'number') {

sum += nextArgument;

}

if (typeof nextArgument === 'string' || nextArgument === null) {

sum += 0;

}

if (!nextArgument) {

return sum;

} else {

return innerCount;

}

}

return innerCount;

}

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

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

```

Результат:

```

18

4

```

**Объекты и массивы Задание 1**

function convertArrToObj(arr) {

const obj = {};

arr.forEach((val, index) => {

obj[index] = val;

});

return obj;

}

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

const obj = convertArrToObj(arr);

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

**Задание 2**

javascript

function countFromArr(arr) {

let res = {};

// Перебираем элементы массива

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

let el = arr[i];

// Если текущий элемент еще не является ключом в объекте res,

// то создаем этот ключ со значением 1

if (!res[el]) {

res[el] = 1;

}

// Иначе, если ключ уже существует, увеличиваем его значение на 1

else {

res[el]++;

}

}

return res;

}

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

const res = countFromArr(arr);

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

**Задание 3**

javascript

function groupByField(arr, field) {

return arr.reduce((result, obj) => {

const key = obj[field];

if (!result[key]) {

result[key] = [];

}

result[key].push(obj);

return result;

}, {});

}

```

Пример использования:

```javascript

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

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

console.log(res);

```

Вывод:

```

{ 1: [{test: 1}, {test: 1}, {test: 1}], 2: [{test: 2}], 3: [{test: 3}]}

**Прототипы, наследование Задание 1**

function plus(number) {

return number + this;

}

Number.prototype.plus = plus;

console.log((1).plus(2)); // 3

**Асинхронная работа в JS” Задание 1**

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

node /tmp/45ml5kUUgx.js

start

promise constructor

final

p2

p4

p5

timeout

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