Git

Команды которые нужно ввести

cd /путь/к/вашему/коду

git init

git add .

git commit -m "Ваше сообщение коммита"

git remote add origin [git@example.com:example/test.git](mailto:git@example.com:example/test.git)

git push -u origin master

--------------------------------------------------------------------------------------------

Css

1. https://codepen.io/igor111122/pen/xxmqvxL

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="styles.css">

<title>Flexbox Layout</title>

</head>

<body>

<header>Header</header>

<nav>Navigation</nav>

<div class="main-content">

<div class="content">Content</div>

<div class="sidebar">Sidebar</div>

</div>

<footer>Footer</footer>

</body>

</html>

body {

display: flex;

flex-direction: column;

height: 100vh;

margin: 0;

}

header,

footer {

flex: 1;

background-color: lightblue;

}

nav {

flex: 0.5;

background-color: lightgray;

}

.main-content {

display: flex;

flex: 4;

}

.content {

flex: 0.6;

background-color: lightgreen;

animation: spinAndChangeColor 4s linear infinite;

}

@keyframes spinAndChangeColor {

0% {

transform: rotate(0deg);

background-color: red;

}

50% {

transform: rotate(180deg);

background-color: orange;

}

100% {

transform: rotate(360deg);

background-color: green;

}

}

.sidebar {

flex: 0.4;

background-color: lightcoral;

}

--------------------------------------------------------------------------------------------

2. https://codepen.io/igor111122/pen/gOZmVOZ

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="styles.css">

<title>CSS Grid Layout</title>

</head>

<body>

<div class="container">

<header class="header">Header</header>

<nav class="navigation">Navigation</nav>

<main class="content">Content</main>

<aside class="sidebar">Sidebar</aside>

<footer class="footer">Footer</footer>

</div>

<div class="animation-container">

<div class="animated-square"></div>

</div>

</body>

</html>

body, html {

height: 100%;

margin: 0;

padding: 0;

}

.container {

display: grid;

grid-template-areas:

"header header"

"navigation navigation"

"content sidebar"

"footer footer";

grid-template-rows: 10% 5% 75% 10%;

grid-template-columns: 60% 40%;

height: 100%;

}

.header {

grid-area: header;

background-color: #3498db;

color: white;

}

.navigation {

grid-area: navigation;

background-color: #e74c3c;

color: white;

}

.content {

grid-area: content;

background-color: #2ecc71;

}

.sidebar {

grid-area: sidebar;

background-color: #f1c40f;

}

.footer {

grid-area: footer;

background-color: #34495e;

color: white;

}

/\* Добавляем стили для контейнера анимации \*/

.animation-container {

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

display: flex;

justify-content: center;

align-items: center;

overflow: hidden;

}

/\* Создаем анимированный квадрат \*/

.animated-square {

width: 100px;

height: 100px;

background-color: red;

animation: rotateColor 5s linear infinite;

}

/\* Создаем ключевую анимацию rotateColor \*/

@keyframes rotateColor {

0% {

transform: rotate(0deg);

background-color: red;

}

50% {

transform: rotate(180deg);

background-color: orange;

}

100% {

transform: rotate(360deg);

background-color: green;

}

}

JS

--------------------------------------------------------------------------------------------

1

function deepEqual(obj1, obj2) {

// Проверяем, являются ли obj1 и obj2 объектами

if (typeof obj1 !== 'object' || typeof obj2 !== 'object') {

return obj1 === obj2; // Если не оба объекта, просто сравниваем их

}

// Проверяем, имеют ли объекты одинаковое количество свойств

const keys1 = Object.keys(obj1);

const keys2 = Object.keys(obj2);

if (keys1.length !== keys2.length) {

return false;

}

// Проверяем каждое свойство объектов рекурсивно

for (const key of keys1) {

if (!keys2.includes(key) || !deepEqual(obj1[key], obj2[key])) {

return false;

}

}

// Если все проверки пройдены, объекты идентичны

return true;

}

console.log(deepEqual({ name: 'test' }, { name: 'test' })); // true

console.log(deepEqual({ name: 'test' }, { name: 'test1' })); // false

console.log(deepEqual({ name: 'test', data: { value: 1 } }, { name: 'test', data: { value: 2 } })); // false

console.log(deepEqual({ name: 'test' }, { name: 'test', age: 10 })); // false

--------------------------------------------------------------------------------------------

2

function chunkArray(arr, chunkSize) {

let index = 0; // Начальный индекс для разделения массива

return {

next: function () {

if (index >= arr.length) {

// Если индекс превышает длину массива, итерация завершена

return { value: undefined, done: true };

}

// Вычисляем следующую часть массива

const chunk = arr.slice(index, index + chunkSize);

// Увеличиваем индекс для следующей итерации

index += chunkSize;

// Возвращаем текущую часть массива и информацию о завершении

return { value: chunk, done: false };

},

};

}

const iterator = chunkArray([1, 2, 3, 4, 5, 6, 7, 8], 3);

console.log(iterator.next()); // { value: [1, 2, 3], done: false }

console.log(iterator.next()); // { value: [4, 5, 6], done: false }

console.log(iterator.next()); // { value: [7, 8], done: false }

console.log(iterator.next()); // { value: undefined, done: true }

--------------------------------------------------------------------------------------------

3

async function bulkRun(functionsAndArgs) {

// Создаем массив для хранения результатов

const results = [];

// Проходим по каждой паре [функция, аргументы]

for (const [func, args] of functionsAndArgs) {

// Создаем Promise для выполнения функции с аргументами

const resultPromise = new Promise((resolve) => {

// Вызываем функцию с аргументами и колбэком для получения результата

func(...args, (result) => {

resolve(result); // Разрешаем Promise с результатом функции

});

});

// Добавляем Promise в массив результатов

results.push(await resultPromise);

}

// Возвращаем массив результатов после выполнения всех функций

return results;

}

const f1 = (cb) => {

cb(1);

};

const f2 = (a, cb) => {

cb(a);

};

const f3 = (a, b, cb) => {

setTimeout(() => cb([a, b]), 1000);

};

bulkRun([

[f1, []],

[f2, [2]],

[f3, [3, 4]],

]).then(console.log);

// Output: [1, 2, [3, 4]]

--------------------------------------------------------------------------------------------

4

function arrayToObject(arr) {

const result = {};

// Проходим по каждому элементу массива

for (const [key, value] of arr) {

// Если значение является массивом, вызываем рекурсивно arrayToObject

if (Array.isArray(value)) {

result[key] = arrayToObject(value);

} else {

result[key] = value;

}

}

return result;

}

var arr = [['name', 'developer'], ['age', 5], ['skills', [['html',4], ['css', 5], ['js',5]]]];

const result = arrayToObject(arr);

console.log(result);

// Output: {

// name: 'developer',

// age: 5,

// skills: {

// html: 4,

// css: 5,

// js: 5

// }

// }

--------------------------------------------------------------------------------------------

5

function objectToArray(obj) {

const result = [];

// Проходим по каждому свойству объекта

for (const key in obj) {

if (obj.hasOwnProperty(key)) {

const value = obj[key];

// Если значение является объектом, вызываем рекурсивно objectToArray

if (typeof value === 'object' && !Array.isArray(value)) {

result.push([key, objectToArray(value)]);

} else {

result.push([key, value]);

}

}

}

return result;

}

const obj = {

name: 'developer',

age: 5,

skills: {

html: 4,

css: 5,

js: 5

}

};

const result = objectToArray(obj);

console.log(result);

// Output: [['name', 'developer'], ['age', 5], ['skills', [['html', 4], ['css', 5], ['js', 5]]]]

--------------------------------------------------------------------------------------------

6

function NotificationException() {}

function ErrorException() {}

function primitiveMultiply(a, b) {

const rand = Math.random();

if (rand < 0.5) {

return a \* b;

} else if (rand > 0.85) {

throw new ErrorException();

} else {

throw new NotificationException();

}

}

function reliableMultiply(a, b) {

while (true) { // Бесконечный цикл

try {

return primitiveMultiply(a, b); // Попытка выполнить умножение

} catch (error) {

if (error instanceof ErrorException) {

throw error; // Если это ErrorException, прокинуть его наверх

} else if (error instanceof NotificationException) {

// Если это NotificationException, продолжить цикл и повторить попытку

continue;

}

}

}

}

try {

console.log(reliableMultiply(8, 8));

} catch (error) {

console.log("Caught an error:", error);

}

--------------------------------------------------------------------------------------------

7

function mapObject(obj, prefix = '') {

const result = {};

for (const key in obj) {

if (obj.hasOwnProperty(key)) {

const fullPath = prefix ? `${prefix}/${key}` : key;

if (typeof obj[key] === 'object' && !Array.isArray(obj[key])) {

// Если значение является объектом, вызываем рекурсивно mapObject

const nestedMap = mapObject(obj[key], fullPath);

Object.assign(result, nestedMap);

} else {

// Если значение не является объектом, добавляем его в результат

result[fullPath] = obj[key];

}

}

}

return result;

}

const obj = {

a: {

b: {

c: 43453,

d: 'My name is Igor'

},

e: [1, 2, 5]

}

};

const result = mapObject(obj);

console.log(result);

// Output: {

// 'a/b/c': 12,

// 'a/b/d': 'Hello World',

// 'a/e': [1, 2, 3]

// }

--------------------------------------------------------------------------------------------

8

function combos(num) {

const result = [];

function findCombos(currentCombo, remainingNum, start) {

if (remainingNum === 0) {

result.push(currentCombo.slice());

return;

}

for (let i = start; i <= remainingNum; i++) {

currentCombo.push(i);

findCombos(currentCombo, remainingNum - i, i);

currentCombo.pop();

}

}

findCombos([], num, 1);

return result;

}

console.log(combos(3));

console.log(combos(10));

--------------------------------------------------------------------------------------------

9

function add(num) {

let sum = num; // Инициализируем сумму первым числом

// Создаем новую функцию, которая ожидает следующее число

const nextAdd = function (nextNum) {

sum += nextNum; // Добавляем число к сумме

return nextAdd; // Возвращаем себя же для возможности следующего вызова

};

// Приводим функцию к числу (когда нужно получить результат)

nextAdd.valueOf = function () {

return sum;

};

return nextAdd; // Возвращаем созданную функцию

}

// Приводим результат к числу

console.log(Number(add(1)(2))); // == 3

console.log(Number(add(1)(2)(5))); // == 8

console.log(Number(add(1)(2)(-3)(4))); // == 4

console.log(Number(add(1)(2)(3)(4)(-5))); // == 5

--------------------------------------------------------------------------------------------