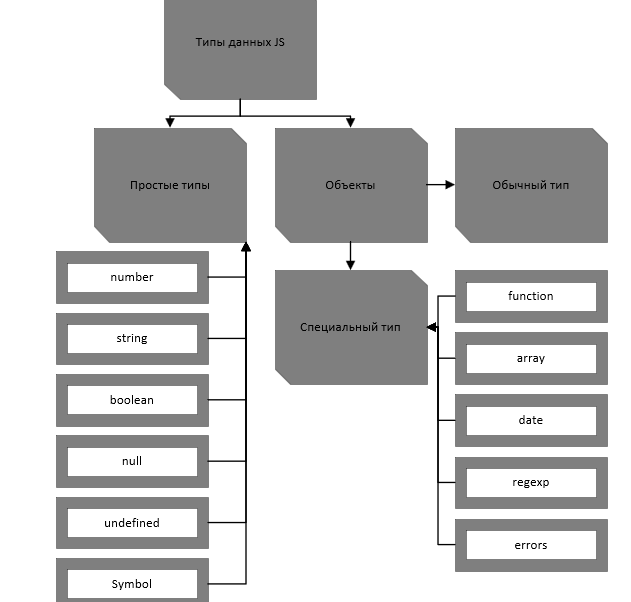
**Задание для ИСП-420п от (23.12.23) (ПИТПМ)**

**Жирков Александр**

1. Основы JavaScript

Диаграмма типов данных



Разница var, let и const

Когда интерпретатор JavaScript находит очередное объявление с var, он выделяет место в памяти и по умолчанию присваивает элементу значение undefined — то есть «неопределённое».

Переменная var, созданная вне функции, действует как глобальная переменная — она доступна из любой части скрипта.

Если же создать переменную с помощью var внутри функции, то она будет локальной, то есть доступной только в этой функции.

Это позволяет создавать переменные с одинаковыми названиями и обращаться к внешним переменным через специальный объект window.

Не стоит привыкать к var. Использование этого ключевого слова в 2022 году считается плохой практикой по нескольким причинам.

1. Отсутствие блочной области видимости.
2. Повторное объявление.
3. «Поднятие» переменной, или hoisting.

Ключевое слово let лишено недостатков своего предшественника. Переменные, объявленные с его помощью, нельзя объявить повторно — программа выдаст ошибку. Let-переменные тоже «всплывают», но при попытке обратиться к ним до инициализации вы получите ошибку ReferenceError.

Но главное — такие переменные имеют блочную область видимости. А значит, они доступны только внутри того блока {}, в котором были созданы.

С помощью ключевого слова const создаются константы. Например, физические и математические величины.

hoisting

Поднятие или hoisting — это механизм в JavaScript, в котором переменные и объявления функций, передвигаются вверх своей области видимости перед тем, как код будет выполнен.

1. Функции

Объявление функции в JS.

function имя(параметры {тело функции}

Разница между function declaration и function expression.

Function Declaration – функция, объявленная в основном потоке кода.

Function Expression – объявление функции в контексте какого-либо выражения, например присваивания.

Стрелочные функции

В JavaScript стрелочная функция — это более короткий синтаксис для написания функциональных выражений. Стрелочные функции — это анонимные функции с более коротким синтаксисом, чем у традиционных функциональных выражений, что делает их идеальными для коротких и простых функций.

1. Объекты и массивы

Создание объекта

Let Имя\_Объекта = new Object();

Добавление нового элемента в конец массива

Метод push(). Массив.push(n);

Переборка всех элементов массива

Методов много, но вот один из них, при помощи метода forEach:

var a = ["a", "b", "c"];

a.forEach(function(entry) {

console.log(entry);

});

1. Обработка событий

Назначение обработчика событий элементу

<input value="Кнопка" onclick="alert('Кнопка')" type="button">

event delegation

Идея в том, что если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому, мы ставим один обработчик на их общего предка.

Отмена стандартного поведения события

Есть два способа отменить действие браузера:

Основной способ – это воспользоваться объектом event. Для отмены действия браузера существует стандартный метод event.preventDefault().

Если же обработчик назначен через on<событие> (не через addEventListener), то также можно вернуть false из обработчика.

1. Асинхронность

Синхронный и асинхронный код в JavaScript.

Выполнение JS-кода — однопоточное. Это значит, что в конкретный момент времени движок может выполнять не более одной строки кода. То есть вторая строка не будет выполнена, пока не выполнится первая.

Такое выполнение кода (строка за строкой) называется синхронным. Асинхронный код – противоположность синхронному, строки будут выполняться дальше, даже если действуют еще предыдущие.

Промисы

Промис — это объект-обёртка для асинхронного кода. Он содержит в себе состояние: вначале pending («ожидание»), затем — одно из: fulfilled («выполнено успешно») или rejected («выполнено с ошибкой»).

async/await

async function f() {

return 1;

}

// работает только внутри async–функций

let value = await promise;

Задания

1. Основы JS

var Name = 'Alexander';

console.log(Name)

var names = ['Alexander1', 'Alexander2', 'Alexander3'];

names.forEach(function(entry) {

    console.log(entry);

});

1. Функции

function summ(a, b) {

return a + b;

}

var a = 1;

var b = 2;

var summa = summ(a, b);

console.log("Сумма:", summa);

function Student(name, age) {

    this.name = name;

    this.age = age;

  }

  var firstStudent = new Student("Александр", 18);

  var secondStudent = new Student("Александр", 20);

  console.log("Имя первого студента:", firstStudent.name);

  console.log("Возраст первого студента:", firstStudent.age);

  console.log("Имя второго студента:", secondStudent.name);

  console.log("Возраст второго студента:", secondStudent.age);

1. Объекты и массивы

var car = {

brand: "Kia",

year: 2023

};

console.log("Марка автомобиля:", car.brand);

console.log("Год выпуска автомобиля:", car.year);

  // Создаем массив

  var numbersArray = [1, 2, 3, 4, 5];

  // Выводим исходный массив

  console.log("Исходный массив:", numbersArray);

  // Вычисляем индекс середины массива

  var middleIndex = Math.floor(numbersArray.length / 2);

  // Добавляем новый элемент в середину массива

  numbersArray.splice(middleIndex, 0, 6);

  // Выводим массив после добавления элемента в середину

  console.log("Массив после добавления:", numbersArray);

1. Асинхронность

function wait() {

return new Promise(function(resolve) {

setTimeout(function() {

resolve('Promise разрешен');

}, 1000);

});

}

wait().then(function(result) {

console.log(result);

});

async function performDelay() {

    try {

      const result = await delay();

      console.log(result);

    } catch (error) {

      console.error('Произошла ошибка:', error);

    }

  }

  function delay() {

    return new Promise(function(resolve) {

      setTimeout(function() {

        resolve('Promise разрешен!');

      }, 1000);

    });

  }

  performDelay();