**Тема "Объекты в JavaScript"**

1. Что понимается под объектом в JavaScript? Приведите примеры известных вам объектов в JavaScript.

В JavaScript объект — это коллекция свойств, где каждое свойство состоит из ключа (имени) и значения. Объекты могут содержать примитивные значения, функции (методы), другие объекты и т.д.

Примеры объектов:

- `window`\*— глобальный объект, представляющий окно браузера.

- `document` — объект, представляющий структуру HTML-документа.

- `Array` — массив, который тоже является объектом.

- `Date` — объект для работы с датой и временем

2. \*\*Приведите пример описания объекта в JS.\*\*

Пример простого объекта, описывающего автомобиль:

javascript

const car = {

brand: 'Toyota',

model: 'Camry',

year: 2020,

color: 'black',

drive: function() {

console.log('Car is driving');

}

}

3. Приведите пример доступа к свойствам объекта в JS.

Доступ к свойствам объекта осуществляется через точечную нотацию или квадратные скобки:

javascript

console.log(car.brand); // 'Toyota'

console.log(car['color']); // 'black'

4. Какая команда в JS позволяет выполнить удаление свойства. Приведите пример.

Для удаления свойства объекта используется оператор `delete`

javascript

delete car.year;

console.log(car.year); // undefined

5. Какая команда в JS позволяет выполнить проверку существования свойства. Приведите пример.

Для проверки существования свойства в объекте можно использовать оператор `in` или метод `hasOwnProperty`:

javascript

console.log('brand' in car); // true

console.log(car.hasOwnProperty('model')); // true

6. Какая команда в JS позволяет выполнить перебор свойств объекта. Приведите пример.

Для перебора всех свойств объекта можно использовать цикл `for...in`:

```javascript

for (let key in car) {

console.log(key + ': ' + car[key]);

}

`

**Тема "ООП в JavaScript"**

1. \*\*В чем заключается суть ООП программирования и его отличие от традиционного процедурного программирования?\*\*

В \*\*объектно-ориентированном программировании (ООП)\*\* основное внимание уделяется объектам — сущностям, которые представляют реальные объекты, их свойства и поведение (методы). В процедурном программировании акцент делается на функциях и алгоритмах, выполняющих действия над данными.

**Основные отличия:**

- ООП организует код вокруг объектов, которые содержат данные (свойства) и методы.

- В процедурном программировании функции и данные отделены, и код выполняется последовательно через вызовы функций.

2. Что понимается под Классом и Объектом в JS?

Класс — это шаблон для создания объектов с общими свойствами и методами. Он определяет, как будет выглядеть объект, его состояние и поведение. \*\*Объект\*\* — это экземпляр класса, представляющий конкретную сущность.

3. \*\*Приведите пример синтаксиса класса в JS.\*\*

Пример класса:

javascript

class Car {

constructor(brand, model) {

this.brand = brand;

this.model = model;

}

drive() {

console.log(`${this.brand} ${this.model} is driving`);

}

}

4. Как осуществляется создание объектов класса? Приведите пример.

Создание объекта класса осуществляется с помощью ключевого слова `new`:

```javascript

const myCar = new Car('Toyota', 'Camry');

myCar.drive(); // 'Toyota Camry is driving'

```

5. Как происходит обращение к полям и методам внутри класса?

Внутри класса обращение к полям и методам происходит через ключевое слово `this`, которое ссылается на текущий объект:

```javascript

class Person {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hello, my name is ${this.name}`);

}

}

const person = new Person('John');

person.greet(); // 'Hello, my name is John'

```

6. Как обозначаются в JS публичные, приватные и защищённые поля и методы?

- Публичные поля и методы доступны везде и обозначаются просто через `this`:

```javascript

this.publicField = value;

```

- \*\*Приватные\*\* поля и методы обозначаются с помощью символа `#` перед именем:

``javascript

#privateField = value;

```

- \*\*Защищённые\*\* поля не поддерживаются стандартом, но можно использовать соглашения, например, начинать имя с символа `\_`.

7. Для чего в ООП в JS используется служебное слово `static`?

Ключевое слово `static` используется для определения методов или свойств класса, которые не зависят от конкретного экземпляра объекта. Такие методы вызываются напрямую на классе:

```javascript

class MathHelper {

static add(a, b) {

return a + b;

}

}

console.log(MathHelper.add(5, 10)); // 15

```

8. Что такое геттеры и сеттеры? Для чего и как они используются в ООП в JS?

\*\*Геттеры\*\* и \*\*сеттеры\*\* используются для управления доступом к свойствам объекта. Геттер позволяет получить значение, а сеттер — изменить его с дополнительной логикой.

Пример:

```javascript

class User {

constructor(name) {

this.\_name = name;

}

get name() {

return this.\_name;

}

set name(newName) {

if (newName.length > 0) {

this.\_name = newName;

}

}

}

const user = new User('Alice');

console.log(user.name); // 'Alice'

user.name = 'Bob';

console.log(user.name); // 'Bob'

```

9. \*\*Что понимается под базовыми принципами ООП: инкапсуляция, наследование и полиморфизм?\*\*

- \*\*Инкапсуляция\*\* — сокрытие данных объекта и предоставление доступа к ним только через публичные методы (геттеры, сеттеры).

- \*\*Наследование\*\* — механизм, позволяющий одному классу наследовать свойства и методы другого класса.

- \*\*Полиморфизм\*\* — способность объектов разных классов реагировать на одинаковые методы по-разному.

10. \*\*Примеры реализации базовых принципов ООП на JS:\*\*

- \*\*Инкапсуляция:\*\*

```javascript

class BankAccount {

#balance = 0;

deposit(amount) {

this.#balance += amount;

}

getBalance() {

return this.#balance;

}

}

const account = new BankAccount();

account.deposit(100);

console.log(account.getBalance()); // 100

```

- \*\*Наследование:\*\*

```javascript

class Animal {

speak() {

console.log('Animal speaks');

}

}

class Dog extends Animal {

speak() {

console.log('Dog barks');

}

}

const dog = new Dog();

dog.speak(); // 'Dog barks'

```

- \*\*Полиморфизм:\*\*

```javascript

class Shape {

draw() {

console.log('Drawing a shape');

}

}

class Circle extends Shape {

draw() {

console.log('Drawing a circle');

}

}

class Square extends Shape {

draw() {

console.log('Drawing a square');

}

}

const shapes = [new Shape(), new Circle(), new Square()];

shapes.forEach(shape => shape.draw());

// 'Drawing a shape', 'Drawing a circle', 'Drawing a square'