Шпаргалки для собеседования Frontend Developer

JavaScript (ES6+)

Основные концепции

Замыкания (Closures)

```javascript
function outer(x) {
  return function inner(y) {
    return x + y; // inner имеет доступ к x из outer
  };
}
const add5 = outer(5);
console.log(add5(10)); // 15
```

Hoisting

```javascript
// Переменные var поднимаются наверх
console.log(x); // undefined (не ошибка!)
var x = 5;

// let/const не инициализируются при подъеме
console.log(y); // ReferenceError
let y = 10;
```

Event Loop и Asynchronous

```javascript
console.log('1');
setTimeout(() => console.log('2'), 0);
Promise.resolve().then(() => console.log('3'));
console.log('4');
// Вывод: 1, 4, 3, 2
```

Контекст выполнения (this)

```javascript
```

```javascript
// В стрелочных функциях this берется из лексического окружения
const obj = {
  name: 'Object',
  regularFunction() {
    console.log(this.name); // 'Object'
  },
  arrowFunction: () => {
    console.log(this.name); // undefined (глобальный контекст)
  }
};
```

Прототипы и наследование

```javascript
function Person(name) {
  this.name = name;
}
Person.prototype.greet = function() {
  return `Hello, ${this.name}`;
};

// ES6 классы - синтаксический сахар над прототипами
class Developer extends Person {
  constructor(name, language) {
    super(name);
    this.language = language;
  }

  code() {
    return `${this.name} codes in ${this.language}`;
  }
}
```

Продвинутые темы

Промисы и async/await

```javascript
```

```javascript
// Promise chaining
fetch('/api/data')
  .then(response => response.json())
  .then(data => processData(data))
  .catch(error => handleError(error));

// Async/await
async function fetchData() {
  try {
    const response = await fetch('/api/data');
    const data = await response.json();
    return processData(data);
  } catch (error) {
    handleError(error);
  }
}

// Promise.all vs Promise.allSettled
const promises = [fetch('/api/1'), fetch('/api/2')];
const results = await Promise.all(promises); // Все должны успешно выполниться
const settled = await Promise.allSettled(promises); // Все результаты, включая ошибки
```

Деструктуризация и Spread/Rest

javascript

```javascript
// Деструктуризация с переименованием и значениями по умолчанию
const { name: userName = 'Anonymous', age } = user;

// Rest в функциях
function sum(first, ...rest) {
  return rest.reduce((acc, num) => acc + num, first);
}

// Spread для клонирования объектов
const newObj = { ...oldObj, updatedField: 'newValue' };
```

Генераторы

javascript

```
function* fibonacci() {
  let [a, b] = [0, 1];
  while (true) {
    yield a;
    [a, b] = [b, a + b];
  }
}

const fib = fibonacci();
console.log(fib.next().value); // 0
console.log(fib.next().value); // 1
```

TypeScript

Основные типы

Примитивные типы

typescript

```
let name: string = "John";
let age: number = 30;
let isActive: boolean = true;
let nothing: null = null;
let notDefined: undefined = undefined;
let id: string | number = "123"; // Union types
```

Объекты и интерфейсы

typescript

```
interface User {
  readonly id: number;
  name: string;
  email?: string; // Опциональное свойство
  [key: string]: any; // Index signature
}

type UserRole = 'admin' | 'user' | 'moderator'; // Literal types

interface Admin extends User {
  role: UserRole;
  permissions: string[];
}
```

Дженерики

```typescript
// Базовые дженерики
function identity<T>(arg: T): T {
  return arg;
}

// Дженерики с ограничениями
interface Lengthwise {
  length: number;
}

function loggingIdentity<T extends Lengthwise>(arg: T): T {
  console.log(arg.length);
  return arg;
}

// Утилитарные типы
interface Todo {
  title: string;
  description: string;
  completed: boolean;
}

type TodoPreview = Pick<Todo, 'title' | 'completed'>;
type TodoInfo = Omit<Todo, 'completed'>;
type PartialTodo = Partial<Todo>;
type RequiredTodo = Required<Todo>;
```

Продвинутые типы

Mapped Types

```typescript
```

```typescript
type Optional<T> = {
  [P in keyof T]?: T[P];
};

type Readonly<T> = {
  readonly [P in keyof T]: T[P];
};

// Conditional types
type NonNullable<T> = T extends null | undefined ? never : T;
```

Template Literal Types

typescript

```typescript
type EventName<T extends string> = `on${Capitalize<T>}`;
type ButtonEvent = EventName<'click'>; // 'onClick'

type ApiPath = `/api/${string}`;
```

React

Основы

Компоненты и JSX

tsx

```tsx
// Функциональный компонент
interface Props {
  title: string;
  children?: React.ReactNode;
}

const Header: React.FC<Props> = ({ title, children }) => {
  return (
    <header>
      <h1>{title}</h1>
      {children}
    </header>
  );
};

// Компонент с дефолтными пропами
const Button = ({
  onClick,
  disabled = false,
  children
}: {
  onClick: () => void;
  disabled?: boolean;
  children: React.ReactNode;
}) => (
  <button onClick={onClick} disabled={disabled}>
    {children}
  </button>
);
```

Хуки

```tsx
```

```tsx
// useState с типизацией
const [count, setCount] = useState<number>(0);
const [user, setUser] = useState<User | null>(null);

// useEffect
useEffect(() => {
  const fetchData = async () => {
    const response = await fetch('/api/data');
    const data = await response.json();
    setData(data);
  };

  fetchData();

  // Cleanup
  return () => {
    // Отмена подписок, таймеров и т.д.
  };
}, [dependency]); // Массив зависимостей

// useCallback для оптимизации
const handleClick = useCallback((id: string) => {
  onItemClick(id);
}, [onItemClick]);

// useMemo для дорогих вычислений
const expensiveValue = useMemo(() => {
  return heavyComputation(data);
}, [data]);
```

Кастомные хуки

```tsx
```

```
function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = useCallback(() => setCount(c => c + 1), []);
  const decrement = useCallback(() => setCount(c => c - 1), []);
  const reset = useCallback(() => setCount(initialValue), [initialValue]);

  return { count, increment, decrement, reset };
}

function useFetch<T>(url: string) {
  const [data, setData] = useState<T | null>(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<Error | null>(null);

  useEffect(() => {
    fetch(url)
      .then(res => res.json())
      .then(setData)
      .catch(setError)
      .finally(() => setLoading(false));
  }, [url]);

  return { data, loading, error };
}
```

Продвинутые паттерны

Context API

tsx

```tsx
interface AppContextType {
  user: User | null;
  login: (user: User) => void;
  logout: () => void;
}

const AppContext = createContext<AppContextType | undefined>(undefined);

export const useAppContext = () => {
  const context = useContext(AppContext);
  if (!context) {
    throw new Error('useAppContext must be used within AppProvider');
  }
  return context;
};

export const AppProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null);

  const login = (user: User) => setUser(user);
  const logout = () => setUser(null);

  return (
    <AppContext.Provider value={{ user, login, logout }}>
      {children}
    </AppContext.Provider>
  );
};
```

Higher-Order Components (HOC)

```tsx
```

```tsx
function withLoading<T extends {}>(
  Component: React.ComponentType<T>
) {
  return function WithLoadingComponent(props: T & { isLoading: boolean }) {
    const { isLoading, ...rest } = props;

    if (isLoading) {
      return <div>Loading...</div>;
    }

    return <Component {...rest as T} />;
  };
}
```

Render Props Pattern

```tsx
interface RenderProps {
  children: (data: any, loading: boolean) => React.ReactNode;
}

const DataFetcher: React.FC<RenderProps> = ({ children }) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  // Логика загрузки данных...

  return <>{children(data, loading)}</>;
};

// Использование:
<DataFetcher>
  {(data, loading) => (
    loading ? <Spinner /> : <DataDisplay data={data} />
  )}
</DataFetcher>
```

Порталы

```tsx
```

```tsx
const Modal: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  return ReactDOM.createPortal(
    <div className="modal-overlay">
      <div className="modal">
        {children}
      </div>
    </div>,
    document.body
  );
};
```

Оптимизация производительности

React.memo и useMemo

```tsx
// React.memo для предотвращения лишних рендеров
const ExpensiveComponent = React.memo<{ data: ComplexData }>(({ data }) => {
  return <div>{/* сложная разметка */}</div>;
}, (prevProps, nextProps) => {
  // Кастомная функция сравнения
  return prevProps.data.id === nextProps.data.id;
});

// useMemo для дорогих вычислений
const processedData = useMemo(() => {
  return data.map(item => ({ ...item, processed: true }));
}, [data]);
```

Lazy Loading и Suspense

```tsx
const LazyComponent = React.lazy(() => import('./LazyComponent'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}
```

Next.js

Основы роутинга

File-based routing (App Router)

```
app/
├── layout.tsx          // Корневой layout
├── page.tsx            // Главная страница (/)
├── about/
│   └── page.tsx        // /about
├── blog/
│   ├── page.tsx        // /blog
│   └── [slug]/
│       └── page.tsx    // /blog/[slug]
└── api/
    └── users/
        └── route.ts    // API endpoint
```

Навигация

tsx

```tsx
import Link from 'next/link';
import { useRouter } from 'next/navigation';

function Navigation() {
  const router = useRouter();

  return (
    <nav>
      <Link href="/about">About</Link>
      <Link href="/blog/first-post">First Post</Link>

      <button onClick={() => router.push('/dashboard')}>
        Go to Dashboard
      </button>
    </nav>
  );
}
```

Рендеринг и загрузка данных

Server Components vs Client Components

tsx

```tsx
// Server Component (по умолчанию в app directory)
async function ServerComponent() {
  const data = await fetch('https://api.example.com/data');
  const posts = await data.json();

  return (
    <div>
      {posts.map(post => (
        <article key={post.id}>{post.title}</article>
      ))}
    </div>
  );
}

// Client Component
'use client';
import { useState, useEffect } from 'react';

function ClientComponent() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('/api/data').then(res => res.json()).then(setData);
  }, []);

  return <div>{data ? JSON.stringify(data) : 'Loading...'}</div>;
}
```

Стратегии рендеринга

```tsx
```

```tsx
// Static Generation (по умолчанию)
export default function StaticPage({ posts }) {
  return <PostList posts={posts} />;
}

// Server-Side Rendering
export async function generateMetadata() {
  return {
    title: 'Dynamic Title',
    description: 'Dynamic description'
  };
}

// Incremental Static Regeneration
export const revalidate = 60; // Revalidate every 60 seconds
```

API Routes

Создание API endpoints

tsx

```tsx
// app/api/users/route.ts
import { NextRequest, NextResponse } from 'next/server';

export async function GET() {
  const users = await db.user.findMany();
  return NextResponse.json(users);
}

export async function POST(request: NextRequest) {
  const body = await request.json();
  const user = await db.user.create({
    data: body,
  });
  return NextResponse.json(user, { status: 201 });
}

// Динамические роуты: app/api/users/[id]/route.ts
export async function GET(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  const user = await db.user.findUnique({
    where: { id: params.id },
  });

  if (!user) {
    return NextResponse.json({ error: 'User not found' }, { status: 404 });
  }

  return NextResponse.json(user);
}
```

Middleware и конфигурация

Middleware

```tsx
```

```tsx
// middleware.ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export function middleware(request: NextRequest) {
  // Аутентификация
  const token = request.cookies.get('token');

  if (!token && request.nextUrl.pathname.startsWith('/dashboard')) {
    return NextResponse.redirect(new URL('/login', request.url));
  }

  // Добавление заголовков
  const response = NextResponse.next();
  response.headers.set('X-Custom-Header', 'custom-value');

  return response;
}

export const config = {
  matcher: ['/dashboard/:path*', '/admin/:path*']
};
```

Частые вопросы на собеседованиях

JavaScript

1. **Разница между `var`, `let`, `const`**
   - `var`: function scope, hoisting с инициализацией undefined
   - `let`/`const`: block scope, hoisting без инициализации (temporal dead zone)
   - `const`: нельзя переопределить, но объекты можно мутировать
2. **Event Loop**
   - Call Stack → Web APIs → Callback Queue → Event Loop → Call Stack
   - Микротаски (Promises) имеют приоритет над макротасками (setTimeout)
3. **Замыкания**
   - Функция + лексическое окружение
   - Сохраняют доступ к переменным внешней функции

React

1. **Жизненный цикл в хуках**

```tsx
```

```
useEffect(() => {
  // componentDidMount + componentDidUpdate
});

useEffect(() => {
  // componentDidMount
}, []);

useEffect(() => {
  return () => {
    // componentWillUnmount
  };
}, []);
```

2. **Когда происходит ре-рендер**
   - Изменение state
   - Изменение props
   - Родительский компонент ре-рендерится
   - Context изменяется
3. **Оптимизация производительности**
   - React.memo для компонентов
   - useCallback для функций
   - useMemo для значений
   - Виртуализация списков
   - Code splitting с React.lazy

TypeScript

1. **Разница между `interface` и `type`**
   - Interface: можно расширять, merge declarations
   - Type: union types, computed properties, более гибкий
2. **Дженерики**
   - Повторное использование типов
   - Ограничения через extends
   - Условные типы

Next.js

1. **SSR vs SSG vs ISR**
   - SSR: рендер на каждый запрос

- SSG: рендер во время билда
- ISR: SSG + revalidation

2. **App Router vs Pages Router**
   - App: новый, server components, layouts
   - Pages: старый, client-side routing

Примеры кода для live coding

Дебаунс хук

tsx

```tsx
function useDebounce<T>(value: T, delay: number): T {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(handler);
    };
  }, [value, delay]);

  return debouncedValue;
}
```

Простой кастомный хук для локального хранилища

tsx

```
function useLocalStorage<T>(key: string, initialValue: T) {
  const [storedValue, setStoredValue] = useState<T>(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      return initialValue;
    }
  });

  const setValue = (value: T | ((val: T) => T)) => {
    try {
      const valueToStore = value instanceof Function ? value(storedValue) : value;
      setStoredValue(valueToStore);
      window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
      console.log(error);
    }
  };

  return [storedValue, setValue] as const;
}
```

Реализация Promise.all

javascript

```
function promiseAll(promises) {
  return new Promise((resolve, reject) => {
    if (promises.length === 0) {
      resolve([]);
      return;
    }

    const results = [];
    let completed = 0;

    promises.forEach((promise, index) => {
      Promise.resolve(promise)
        .then(result => {
          results[index] = result;
          completed++;
          if (completed === promises.length) {
            resolve(results);
          }
        })
        .catch(reject);
    });
  });
}
```