**✅ 1. CSS-переменные (custom properties)**

**📌 Объявляются глобально и используются в любом компоненте**

**Пример:**

**📄 index.css или App.css:**

:root {

--main-color: #D84343;

--font-size-base: 16px;

--max-width: 1200px;

--button-radius: 8px;

}

**В React-компоненте (через sx, style или CSS):**

<Box

sx={{

color: 'var(--main-color)',

fontSize: 'var(--font-size-base)',

maxWidth: 'var(--max-width)',

borderRadius: 'var(--button-radius)',

}}

>

Пример с CSS-переменными

</Box>

✅ **Плюсы:** работает в любом компоненте, легко обновлять дизайн.  
❌ **Минусы:** не даёт автодополнения, типизация отсутствует.

**✅ 2. Темы через MUI ThemeProvider (если ты используешь Material UI)**

**📌 Создаёшь глобальную тему с цветами, шрифтами, размерами и используешь через sx или theme.**

**Пример:**

**📄 theme.js:**

import { createTheme } from '@mui/material/styles';

const theme = createTheme({

palette: {

primary: {

main: '#D84343',

},

text: {

main: '#101828',

},

},

typography: {

fontSize: 16,

fontFamily: 'Roboto, sans-serif',

},

shape: {

borderRadius: 8,

},

custom: {

maxWidth: '1200px',

},

});

export default theme;

**📄 App.jsx:**

import { ThemeProvider } from '@mui/material/styles';

import theme from './theme';

import YourComponents from './YourComponents';

function App() {

return (

<ThemeProvider theme={theme}>

<YourComponents />

</ThemeProvider>

);

}

**📄 Использование в компоненте:**

import { Box, useTheme } from '@mui/material';

const MyComponent = () => {

const theme = useTheme();

return (

<Box

sx={{

color: theme.palette.primary.main,

fontSize: theme.typography.fontSize,

borderRadius: theme.shape.borderRadius,

maxWidth: theme.custom.maxWidth,

}}

>

Компонент с глобальной темой

</Box>

);

};

✅ **Плюсы:** автодополнение, централизованный контроль, совместимость с MUI  
✅ **Можно добавлять кастомные поля** (как theme.custom.maxWidth)  
❌ **Немного больше кода**

**✅ 3. Отдельный JS-модуль с переменными**

**📌 Когда ты не используешь MUI или CSS, можно просто вынести переменные в файл:**

**📄 designTokens.js:**

export const colors = {

primary: '#D84343',

text: '#101828',

};

export const sizes = {

fontBase: '16px',

maxWidth: '1200px',

borderRadius: '8px',

};

**Как правильно: использовать предопределённый объект стилей**

Если у тебя есть глобальный объект вроде:

// styleTokens.js

export const globalText = {

fontSize: '16px',

lineHeight: 1.5,

};

Тогда в компоненте ты можешь просто **распространить (...)** его внутрь sx:

import { globalText } from '../styleTokens';

<Typography sx={{ ...globalText, color: 'red' }}>

Текст с глобальным стилем

</Typography>

✅ Так ты переиспользуешь переменные корректно.

**✳️ Бонус: можно сделать функцию**

Если глобальные стили часто зависят от условий, ты можешь сделать функцию:

export const getGlobalTextStyle = (isHighlighted) => ({

fontSize: '16px',

lineHeight: 1.5,

color: isHighlighted ? '#D84343' : '#101828',

});

И использовать так:

<Typography sx={getGlobalTextStyle(true)}>

Акцентированный текст

</Typography>

**🧠 Вывод**

* ✅ Используй sx={{ ...globalText }} — работает корректно.
* ❌ sx={{ global }} — не сработает, потому что sx не понимает "глобальные блоки".
* ✅ Хорошая практика — хранить стили в отдельных модулях (типа styleTokens.js или theme.js).

**Использование:**

import { colors, sizes } from '../designTokens';

<Box

sx={{

color: colors.primary,

fontSize: sizes.fontBase,

maxWidth: sizes.maxWidth,

borderRadius: sizes.borderRadius,

}}

>

Компонент с переменными из JS

</Box>

✅ **Плюсы:** просто, можно использовать в логике (не только в стилях)  
❌ **Не работает в чистом CSS (только в JS)**