Font Optimization

`next/font` will automatically optimize your fonts (including custom fonts) and remove external network requests for improved privacy and performance.

> **Watch:** Learn more about how to use `next/font` → YouTube (6 minutes)↗.

`next/font` includes **built-in automatic self-hosting** for *any* font file. This means you can optimally load web fonts with zero layout shift, thanks to the underlying CSS `size-adjust` property used.

This new font system also allows you to conveniently use all Google Fonts with performance and privacy in mind. CSS and font files are downloaded at build time and self-hosted with the rest of your static assets. **No requests are sent to Google by the browser.**

Google Fonts

Automatically self-host any Google Font. Fonts are included in the deployment and served from the same domain as your deployment. **No requests are sent to Google by the browser.**

Get started by importing the font you would like to use from `next/font/google` as a function. We recommend using variable fonts↗ for the best performance and flexibility.

To use the font in all your pages, add it to `_app.js` file under `/pages` as shown below:

pages/_app.js

```
import { Inter } from 'next/font/google';

// If loading a variable font, you don't need to specify the font weight
const inter = Inter({ subsets: ['latin'] });

export default function MyApp({ Component, pageProps }) {
```

```
  return (
    <main className={inter.className}>
      <Component {...pageProps} />
    </main>
  );
}
```

If you can't use a variable font, you will **need to specify a weight**:

pages/_app.js

```
import { Roboto } from 'next/font/google';

const roboto = Roboto({
  weight: '400',
  subsets: ['latin'],
});

export default function MyApp({ Component, pageProps }) {
  return (
    <main className={roboto.className}>
      <Component {...pageProps} />
    </main>
  );
}
```

You can specify multiple weights and/or styles by using an array:

app/layout.js

```
const roboto = Roboto({
  weight: ['400', '700'],
  style: ['normal', 'italic'],
  subsets: ['latin'],
  display: 'swap',
});
```

Good to know: Use an underscore (_) for font names with multiple words. E.g. `Roboto Mono` should be imported as `Roboto_Mono`.

Apply the font in `<head>`

You can also use the font without a wrapper and `className` by injecting it inside the `<head>` as follows:

pages/_app.js

```
import { Inter } from 'next/font/google';

const inter = Inter({ subsets: ['latin'] });

export default function MyApp({ Component, pageProps }) {
  return (
    <>
      <style jsx global>{`
        html {
          font-family: ${inter.style.fontFamily};
        }
      `}</style>
      <Component {...pageProps} />
    </>
  );
}
```

Single page usage

To use the font on a single page, add it to the specific page as shown below:

pages/index.js

```
import { Inter } from 'next/font/google';

const inter = Inter({ subsets: ['latin'] });

export default function Home() {
  return (
    <div className={inter.className}>
      <p>Hello World</p>
    </div>
  );
}
```

Specifying a subset

Google Fonts are automatically subset↗. This reduces the size of the font file and improves performance. You'll need to define which of these subsets you want to preload. Failing to specify any subsets while `preload` is `true` will result in a warning.

This can be done by adding it to the function call:

pages/_app.js

```
const inter = Inter({ subsets: ['latin'] });
```

View the Font API Reference for more information.

Using Multiple Fonts

You can import and use multiple fonts in your application. There are two approaches you can take.

The first approach is to create a utility function that exports a font, imports it, and applies its `className` where needed. This ensures the font is preloaded only when it's rendered:

app/fonts.ts

```ts
import { Inter, Roboto_Mono } from 'next/font/google';

export const inter = Inter({
  subsets: ['latin'],
  display: 'swap',
});

export const roboto_mono = Roboto_Mono({
  subsets: ['latin'],
  display: 'swap',
});
```

In the example above, `Inter` will be applied globally, and `Roboto Mono` can be imported and applied as needed.

Alternatively, you can create a CSS variable and use it with your preferred CSS solution:

app/global.css

```css
html {
  font-family: var(--font-inter);
}

h1 {
  font-family: var(--font-roboto-mono);
}
```

In the example above, `Inter` will be applied globally, and any `<h1>` tags will be styled with `Roboto Mono`.

Recommendation: Use multiple fonts conservatively since each new font is an additional resource the client has to download.

Local Fonts

Import `next/font/local` and specify the `src` of your local font file. We recommend using variable fonts ↗ for the best performance and flexibility.

pages/_app.js

```
import localFont from 'next/font/local';

// Font files can be colocated inside of `pages`
const myFont = localFont({ src: './my-font.woff2' });

export default function MyApp({ Component, pageProps }) {
  return (
    <main className={myFont.className}>
      <Component {...pageProps} />
    </main>
  );
}
```

If you want to use multiple files for a single font family, `src` can be an array:

```
const roboto = localFont({
  src: [
    {
      path: './Roboto-Regular.woff2',
      weight: '400',
      style: 'normal',
    },
    {
      path: './Roboto-Italic.woff2',
      weight: '400',
      style: 'italic',
    },
    {
      path: './Roboto-Bold.woff2',
      weight: '700',
      style: 'normal',
    },
    {
      path: './Roboto-BoldItalic.woff2',
      weight: '700',
      style: 'italic',
    },
  ],
});
```

View the Font API Reference for more information.

With Tailwind CSS

`next/font` can be used with Tailwind CSS ↗ through a CSS variable.

In the example below, we use the font `Inter` from `next/font/google` (you can use any font from Google or Local Fonts). Load your font with the `variable` option to define your CSS variable name and assign it to `inter`. Then, use `inter.variable` to add the CSS variable to your HTML document.

pages/_app.js

```
import { Inter } from 'next/font/google';

const inter = Inter({
  subsets: ['latin'],
  variable: '--font-inter',
});

export default function MyApp({ Component, pageProps }) {
  return (
    <main className={`${inter.variable} font-sans`}>
      <Component {...pageProps} />
    </main>
  );
}
```

Finally, add the CSS variable to your Tailwind CSS config:

tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    './pages/**/*.{js,ts,jsx,tsx}',
    './components/**/*.{js,ts,jsx,tsx}',
  ],
  theme: {
    extend: {
      fontFamily: {
        sans: ['var(--font-inter)'],
        mono: ['var(--font-roboto-mono)'],
      },
    },
  },
  plugins: [],
```

```
16  };
```

You can now use the `font-sans` and `font-mono` utility classes to apply the font to your elements.

Preloading

When a font function is called on a page of your site, it is not globally available and preloaded on all routes. Rather, the font is only preloaded on the related route/s based on the type of file where it is used:

- if it's a unique page, it is preloaded on the unique route for that page
- if it's in the custom App, it is preloaded on all the routes of the site under `/pages`

Reusing fonts

Every time you call the `localFont` or Google font function, that font is hosted as one instance in your application. Therefore, if you load the same font function in multiple files, multiple instances of the same font are hosted. In this situation, it is recommended to do the following:

- Call the font loader function in one shared file
- Export it as a constant
- Import the constant in each file where you would like to use this font