requests:

```
const puppeteer = require('puppeteer');

(async () => {

  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.setRequestInterception(true);
  page.on('request', (interceptedRequest) => {
    if (interceptedRequest.isInterceptResolutionHandle
    if (
      interceptedRequest.url().endsWith('.png') ||
      interceptedRequest.url().endsWith('.jpg')
    )
      interceptedRequest.abort();
    else interceptedRequest.continue();
  });
  await page.goto('https://example.com');
  await browser.close();
})();
```

Multiple Intercept Handlers and Asynchronous Resolutions

By default Puppeteer will raise a `Request is already handled!` exception if `request.abort`, `request.continue`, or `request.respond` are called after any of them have already been called.

Always assume that an unknown handler may have already called `abort/continue/respond`. Even if your handler is the only one you registered, 3rd party packages may register their own handlers. It is therefore important to always check the resolution status using request.isInterceptResolutionHandled before calling `abort/continue/respond`.

Importantly, the intercept resolution may get handled by another listener while your handler is awaiting an asynchronous operation. Therefore, the return value of `request.isInterceptResolutionHandled` is only safe in a synchronous code block. Always execute `request.isInterceptResolutionHandled` and `abort/continue/respond` **synchronously** together.

This example demonstrates two synchronous handlers working together:

```
/*
This first handler will succeed in calling request.con
```