4.2 command-line-arguments.ts

```
/**
 * @title Command Line Arguments
 * @difficulty beginner
 * @tags cli
 * @run <url> Deno Sushi --help --version=1.0.0 --no-color
 * @resource {https://doc.deno.land/deno/stable/~/Deno.args} Doc:
Deno.args
 * @resource
{https://doc.deno.land/https://deno.land/std@0.120.0/flags/mod.ts}
Doc: std/flags
 *
 * Command line arguments are often used to pass configuration options
to a
 * program.
 */

// You can get the list of command line arguments from `Deno.args`.
const name = Deno.args[0];
const food = Deno.args[1];
console.log(`Hello ${name}, I like ${food}!`);

// Often you want to parse command line arguments like `--foo=bar`
into
// structured data. This can be done using `std/flags`.
import { parse } from "https://deno.land/std@0.119.0/flags/mod.ts";

// The `parse` function takes the argument list, and a list of
options. In these
// options you specify the types of the accepted arguments and
possibly default
// values. An object is returned with the parsed arguments.
const flags = parse(Deno.args, {
  boolean: ["help", "color"],
  string: ["version"],
  default: { color: true },
});
console.log("Wants help?", flags.help);
console.log("Version:", flags.version);
console.log("Wants color?:", flags.color);

// The `_` field of the returned object contains all arguments that
were not
// recognized as flags.
console.log("Other:", flags._);
```

4.3 create-remove-directories.ts

```
/**
 * @title Creating & Removing Directories
```


1 /

1.1 CODEOWNERS

```
*   @lucacasonato
```

1.2 deno.jsonc

```
{
  "tasks": {
    "dev": "deno run --allow-net --allow-read=. --allow-env --no-check
--watch=data www/main.ts",
    "fmt": "deno fmt",
    "fmt_check": "deno fmt --check",
    "lint": "deno lint",
    "test": "deno test -A"
  }
}
```

1.3 toc.js

```
export const TOC = [
  "hello-world",
  "color-logging",
  "import-export",
  "dependency-management",
  "importing-json",
  "parsing-serializing-json",
  "timers",
  "prompts",
  "deno-version",
  "pid",
  "environment-variables",
  "command-line-arguments",
  "reading-files",
  "writing-files",
  "moving-renaming-files",
  "temporary-files",
  "create-remove-directories",
  "http-requests",
  "uuids",
  "http-server",
  "http-server-routing",
  "http-server-streaming",
];
```

2 .github/

2.1 workflows/ci.yml

```yaml
name: ci

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  format_lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up Deno
        uses: denoland/setup-deno@v1.0.0

      - name: Format
        run: deno task fmt_check

      - name: Format
        run: deno task lint

      - name: Test
        run: deno task test
```

3 .vscode/

3.1 settings.json

```json
{
  "deno.enable": true,
  "deno.lint": true,
  "deno.unstable": false
}
```


4 data/

4.1 color-logging.ts

```ts
/**
 * @title Logging with colors
 * @difficulty beginner
 * @tags cli, deploy, web
 * @run <url>
 * @resource {https://developer.mozilla.org/en-US/docs/Web/API/console#styling_console_output} MDN: Styling console output
 *
 * Most modern terminals can display program logs in colors and with text
 * decorations. This example shows how to display colors when using
 * `console.log`.
 */

// Deno has support for CSS using the %c syntax in console.log. Here, the text
// "Hello World" is displayed in red. This also works in the browser.
console.log("%cHello World", "color: red");

// Not just foreground, but also background colors can be set.
console.log("%cHello World", "background-color: blue");

// Text decorations like underline and strikethrough can be set too.
console.log("%cHello World", "text-decoration: underline");
console.log("%cHello World", "text-decoration: line-through");

// Font weight can also be set (either to normal, or to bold).
console.log("%cHello World", "font-weight: bold");

// Multiple CSS rules can also be applied at once. Here the text is red and bold.
console.log("%cHello World", "color: red; font-weight: bold");

// A single console.log can also contain multiple %c values. Styling is reset at
// every %c.
console.log("%cHello %cWorld", "color: red", "color: blue");

// Instead of predefined colors, hex literals and `rgb()` are also supported.
// Note that some terminals do not support displaying these colors.
console.log("%cHello World", "color: #FFC0CB");
console.log("%cHello World", "color: rgb(255, 192, 203)");

// It is not possible to configure font size, font family, leading, or any other
// CSS attributes.
```

```
 * @resource {https://deno.land/#installation} Deno: Installation
 * @resource
{https://deno.land/manual@v1.17.2/getting_started/setup_your_environme
nt} Manual: Set up your environment
 */

// The one and only line in this program will print "Hello, World!" to
the
// console.
console.log("Hello, World!");

// Deno programs can either be written in JavaScript or TypeScript, or
a mixture
// of both. All code in these examples is written in TypeScript, but
all the
// examples also work in JavaScript.
```

4.8 http-requests.ts

```
/**
 * @title HTTP Requests
 * @difficulty beginner
 * @tags cli, deploy, web
 * @run --allow-net <url>
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Fetch_API} MDN: Fetch API
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/ReadableStream} MDN: ReadableStream
 *
 * This example demonstrates how to make a HTTP request to a server.
 */

// To make a request to a server, you use the `fetch` API.
let resp = await fetch("https://example.com");

// The response is a `Response` object. This contains the status code,
headers,
// and the body.
console.log(resp.status); // 200
console.log(resp.headers.get("Content-Type")); // "text/html"
console.log(await resp.text()); // "Hello, World!"

// The response body can also be read as JSON, an ArrayBuffer, or a
Blob. A body
// can be read only once.
resp = await fetch("https://example.com");
await resp.arrayBuffer();
/** or await response2.json(); */
/** or await response2.blob(); */
```



```
 * @difficulty beginner
 * @tags cli
 * @run --allow-read --allow-write <url>
 * @resource {https://doc.deno.land/deno/stable/~/Deno.mkdir} Doc:
Deno.mkdir
 * @resource {https://doc.deno.land/deno/stable/~/Deno.remove} Doc:
Deno.remove
 *
 * Creating and removing directories is a common task. Deno has a
number of
 * functions for this task.
 */

// The `Deno.mkdir()` function creates a directory at the specified
path.
// If the directory already exists, it errors.
await Deno.mkdir("new_dir");

// A directory can also be created recursively. In the code below,
three new
// directories are created: `./dir`, `./dir/dir2`, and
`./dir/dir2/subdir`. If
// the recursive option is specified the function will not error if
any of the
// directories already exist.
await Deno.mkdir("./dir/dir2/subdir", { recursive: true });

// Directories can also be removed. This function below removes the
`./new_dir`
// directory. If the directory is not empty, the function will error.
await Deno.remove("./new_dir");

// To remove a directory recursively, use the `recursive` option. This
will
// remove the `./dir` directory and all of its contents.
await Deno.remove("./dir", { recursive: true });

// Synchronous versions of the above functions are also available.
Deno.mkdirSync("new_dir");
Deno.removeSync("new_dir");

// Creating and removing directories requires the `read` and `write`
// permissions.
```

4.4 deno-version.ts

```
/**
 * @title Getting the Deno version
 * @difficulty beginner
```

```
 * @tags cli
 * @run <url>
 * @resource {https://doc.deno.land/deno/stable/~/Deno.version} Doc:
Deno.version
 *
 * How to examine the version of Deno being used.
 */

// To print the current version of Deno, just reach into the Deno
global object
// where all non-web-standard APIs reside.
console.log("Current Deno version", Deno.version.deno);

// Deno has two main dependencies: the V8 JavaScript engine (from the
Chrome web
// browser) and the TypeScript compiler. The versions of these are
also
// accessible in the `Deno.version` object.
console.log("Current TypeScript version", Deno.version.typescript);
console.log("Current V8 version", Deno.version.v8);
```

4.5 dependency-management.ts

```
/**
 * @title Dependency Management
 * @difficulty beginner
 * @tags cli, deploy
 * @resource {/import-export} Example: Importing & Exporting
 *
 * It is unwieldy to have to import the same remote module over and over again.
 * Deno provides some conventions to make managing dependencies easier.
 */

// File: ./deps.ts

// The Deno ecosystem has a convention to re-export all remote dependencies from
// a deps.ts file at the root of the repo. This keeps remote dependencies
// organized, and in a single place.
export * as http from "https://deno.land/std@0.119.0/http/mod.ts";
export * as path from "https://deno.land/std@0.119.0/path/mod.ts";

// File: ./main.ts

// Other files can then import dependencies from the deps.ts file.
// deno-lint-ignore no-unused-vars
import { path } from "./deps.ts";

// Doing this makes package version upgrades really easy, as all external
// dependency specifiers live in the same file.
```

4.6 environment-variables.ts

```
/**
 * @title Environment Variables
 * @difficulty beginner
 * @tags cli, deploy
 * @run --allow-env <url>
```



```
 * @resource {https://doc.deno.land/deno/stable/~/Deno.env} Doc: Deno.env
 * @resource {https://deno.com/deploy/docs/projects#environment-variables} Deploy
Docs: Environment Variables
 *
 * Environment variables can be used to configure the behavior of a program,
 * or pass data from one program to another.
 */

// Here an environment variable with the name "PORT" is read. If this
variable
// is set the return value will be a string. If it is unset it will be
`undefined`.
const PORT = Deno.env.get("PORT");
console.log("PORT:", PORT);

// You can also get an object containing all environment variables.
const env = Deno.env.toObject();
console.log("env:", env);

// Environment variables can also be set. The set environment variable
only affects
// the current process, and any new processes that are spawned from
it. It does
// not affect parent processes or the user shell.
Deno.env.set("MY_PASSWORD", "123456");

// You can also unset an environment variable.
Deno.env.delete("MY_PASSWORD");

// Note that environment variables are case-sensitive on unix, but not
on
// Windows. This means that these two invocations will have different
results
// between platforms.
Deno.env.set("MY_PASSWORD", "123");
Deno.env.set("my_password", "456");
console.log("UPPERCASE:", Deno.env.get("MY_PASSWORD"));
console.log("lowercase:", Deno.env.get("my_password"));

// Access to environment variables is only possible if the Deno process is
// running with env var permissions (`--allow-env`). You can limit the permission
// to only a specific number of environment variables (`--allow-
env=PORT,MY_PASSWORD`).
```

4.7 hello-world.ts

```
/**
 * @title Hello World
 * @difficulty beginner
 * @tags cli, deploy, web
 * @run <url>
```

```
  // Return a response with the stream as the body.
  return new Response(body, {
    headers: {
      "content-type": "text/plain",
      "x-content-type-options": "nosniff",
    },
  });
}

// To start the server on the default port, call `serve` with the
handler.
console.log("Listening on http://localhost:8000");
serve(handler);
```

4.11 http-server.ts

```
/**
 * @title HTTP Server: Hello World
 * @difficulty intermediate
 * @tags cli, deploy
 * @run --allow-net <url>
 * @resource {https://doc.deno.land/https://deno.land/std/http/mod.ts} Doc: std/http
 * @resource {https://developer.mozilla.org/en-US/docs/Web/API/Response} MDN:
Response
 * @playground https://dash.deno.com/playground/example-helloworld
 *
 * An example of a HTTP server that serves a "Hello World" message.
 */

// Import the http server from std/http.
import { serve } from "https://deno.land/std@0.119.0/http/server.ts";

// HTTP servers need a handler function. This function is called for every
// request that comes in. It must return a `Response`. The handler function can
// be asynchronous (it may return a `Promise`).
function handler(_req: Request): Response {
  return new Response("Hello, World!");
}

// To start the server on the default port, call `serve` with the handler.
console.log("Listening on http://localhost:8000");
serve(handler);
```

4.12 import-export.ts

```
/**
 * @title Importing & Exporting
 * @difficulty beginner
 * @tags cli, deploy
 * @resource {/dependency-management} Example: Dependency Management
 * @resource
{https://deno.land/manual@v1.17.2/linking_to_external_code} Manual:
Linking to third party code
```



```
// The response body can also be streamed in chunks.
resp = await fetch("https://example.com");
for await (const chunk of resp.body!) {
  console.log("chunk", chunk);
}

// When making a request, you can also specify the method, headers,
and a body.
const body = `{"name": "Deno"}`;
resp = await fetch("https://example.com", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "X-API-Key": "foobar",
  },
  body,
});

// `fetch` also accepts a `Request` object instead of URL + options.
const req = new Request("https://example.com", {
  method: "DELETE",
});
resp = await fetch(req);

// Instead of a string, the body can also be any typed array, blob, or
// a URLSearchParams object.
const url = "https://example.com";
new Request(url, {
  method: "POST",
  body: new Uint8Array([1, 2, 3]),
});
new Request(url, {
  method: "POST",
  body: new Blob(["Hello, World!"]),
});
new Request(url, {
  method: "POST",
  body: new URLSearchParams({ "foo": "bar" }),
});

// Forms can also be sent with `fetch` by using a `FormData` object as
the body.
const formData = new FormData();
formData.append("name", "Deno");
formData.append("file", new Blob(["Hello, World!"]), "hello.txt");
resp = await fetch("https://example.com", {
  method: "POST",
  body: formData,
```

```
});

// Fetch also supports streaming the request body.
const bodyStream = new ReadableStream({
  start(controller) {
    controller.enqueue("Hello, World!");
    controller.close();
  },
});
resp = await fetch("https://example.com", {
  method: "POST",
  body: bodyStream,
});
```

4.9 http-server-routing.ts

```
/**
 * @title HTTP Server: Routing
 * @difficulty intermediate
 * @tags cli, deploy
 * @run --allow-net <url>
 * @resource {/http-server} Example: HTTP Server: Hello World
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/URL_Pattern_API} MDN: URL Pattern API
 * @playground https://dash.deno.com/playground/example-routing
 *
 * An example of a HTTP server that handles requests with different
responses
 * based on the incoming URL.
 */

// Import the http server from std/http.
import { serve } from "https://deno.land/std@0.114.0/http/server.ts";

// URL patterns can be used to match request URLs. They can contain
named groups
// that can be used to extract parts of the URL, e.g. the book ID.
const BOOK_ROUTE = new URLPattern({ pathname: "/books/:id" });

function handler(req: Request): Response {
  // Match the incoming request against the URL patterns.
  const match = BOOK_ROUTE.exec(req.url);
  // If there is a match, extract the book ID and return a response.
  if (match) {
    const id = match.pathname.groups.id;
    return new Response(`Book ${id}`);
  }

  // If there is no match, return a 404 response.
  return new Response("Not found (try /books/1)", {
    status: 404,
  });
}

// To start the server on the default port, call `serve` with the
handler.
console.log("Listening on http://localhost:8000");
serve(handler);
```

4.10 http-server-streaming.ts

```
/**
 * @title HTTP Server: Streaming
 * @difficulty intermediate
 * @tags cli, deploy
 * @run --allow-net <url>
 * @resource {/http-server} Example: HTTP Server: Hello World
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/ReadableStream} MDN: ReadableStream
 * @playground https://dash.deno.com/playground/example-streaming
 *
 * An example HTTP server that streams a response back to the client.
 */

// Import the http server from std/http.
import { serve } from "https://deno.land/std@0.114.0/http/server.ts";

function handler(_req: Request): Response {
  // Set up a variable to store a timer ID, and the ReadableStream.
  let timer: number | undefined = undefined;
  const body = new ReadableStream({
    // When the stream is first created, start an interval that will
emit a
    // chunk every second containing the current time.
    start(controller) {
      timer = setInterval(() => {
        const message = `It is ${new Date().toISOString()}\n`;
        controller.enqueue(new TextEncoder().encode(message));
      }, 1000);
    },
    // If the stream is closed (the client disconnects), cancel the
interval.
    cancel() {
      if (timer !== undefined) {
        clearInterval(timer);
      }
    },
  });
```

```
};
const json = JSON.stringify(obj);
console.log(json);
//- {"hello":"world","numbers":[1,2,3]}

// By default JSON.stringify will output a minified JSON string. You
can
// customize this by specifying an indentation number in the third
argument.
const json2 = JSON.stringify(obj, null, 2);
console.log(json2);
//- {
//-   "hello": "world",
//-   "numbers": [
//-     1,
//-     2,
//-     3
//-   ]
//- }
```

4.16 pid.ts

```
/**
 * @title Process Information
 * @difficulty beginner
 * @tags cli
 * @run <url>
 */

// The current process's process ID is available in the `Deno.pid`
variable.
console.log(Deno.pid);

// The parent process ID is available in the Deno namespace too.
console.log(Deno.ppid);
```

4.17 prompts.ts

```
/**
 * @title Input Prompts
 * @difficulty beginner
 * @tags cli, web
 * @run <url>
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Window/prompt} MDN: prompt
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Window/alert} MDN: alert
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Window/confirm} MDN: confirm
 *
 * Prompts are used to ask the user for input or feedback on actions.
```



```
 *
 * To build composable programs, it is necessary to be able to import
and export
 * functions from other modules. This is accomplished by using ECMA
script
 * modules in Deno.
 */

// File: ./util.ts

// To export a function, you use the export keyword.
export function sayHello(thing: string) {
  console.log(`Hello, ${thing}!`);
}

// You can also export types, variables, and classes.
// deno-lint-ignore no-empty-interface
export interface Foo {}
export class Bar {}
export const baz = "baz";

// File: ./main.ts

// To import things from files other files can use the import keyword.
import { sayHello } from "./util.ts";
sayHello("World");

// You can also import all exports from a file.
import * as util from "./util.ts";
util.sayHello("World");

// Imports don't have to be relative, they can also reference absolute
file or
// https URLs.
import { VERSION } from "https://deno.land/std/version.ts";
console.log(VERSION);
```

4.13 importing-json.ts

```
/**
 * @title Importing JSON
 * @difficulty beginner
 * @tags cli, web
 *
 * JSON files can be imported in JS and TS files using the `import`
keyword.
 * This makes including static data in a library much easier.
 */
```

```ts
// File: ./main.ts

// JSON files can be imported in JS and TS modules. When doing so, you need to
// specify the "json" import assertion type.
import file from "./version.json" assert { type: "json" };
console.log(file.version);

// Dynamic imports are also supported.
const module = await import("./version.json", {
  assert: { type: "json" },
});
console.log(module.default.version);

/* File: ./version.json
{
  "version": "1.0.0"
}
*/
```

4.14 moving-renaming-files.ts

```ts
/**
 * @title Moving/Renaming Files
 * @difficulty beginner
 * @tags cli
 * @run --allow-read=./ --allow-write=./ <url>
 * @resource {https://doc.deno.land/deno/stable/~/Deno.rename} Doc: Deno.rename
 *
 * An example of how to move and rename files and directories in Deno.
 */

// To rename or move a file, you can use the `Deno.rename` function. The first
// argument is the path to the file to rename. The second argument is the new
// path.
await Deno.writeTextFile("./hello.txt", "Hello World!");
await Deno.rename("./hello.txt", "./hello-renamed.txt");
console.log(await Deno.readTextFile("./hello-renamed.txt"));

// If the source file or the destination directory does not exist, the function
// will reject the returned promise with a `Deno.errors.NotFound` error. You can
// catch this error with a `try/catch` block.
try {
  await Deno.rename("./hello.txt", "./does/not/exist");
} catch (err) {
```



```ts
  console.error(err);
}

// A synchronous version of this function is also available.
Deno.renameSync("./hello-renamed.txt", "./hello-again.txt");

// If the destination file already exists, it will be overwritten.
await Deno.writeTextFile("./hello.txt", "Invisible content.");
await Deno.rename("./hello-again.txt", "./hello.txt");
console.log(await Deno.readTextFile("./hello.txt"));

// Read and write permissions are necessary to perform this operation. The
// source file needs to be readable and the destination path needs to be
// writable.
```

4.15 parsing-serializing-json.ts

```ts
/**
 * @title Parsing and serializing JSON
 * @difficulty beginner
 * @tags cli, deploy, web
 * @run <url>
 * @resource {https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON} MDN: JSON
 *
 * JSON is a widely used data interchange format. It is a human-readable, but
 * also easially machine-readable.
 */

// To parse a JSON string, you can use the builtin JSON.parse function. The
// value is returned as a JavaScript object.
const text = `{
  "hello": "world",
  "numbers": [1, 2, 3]
}`;
const data = JSON.parse(text);
console.log(data.hello);
console.log(data.numbers.length);

// To turn a JavaScript object into a JSON string, you can use the builtin
// JSON.stringify function.
const obj = {
  hello: "world",
  numbers: [1, 2, 3],
```

```
console.log("Temp dir path 2:", tempDirPath2);

// Synchronous versions of the above functions are also available.
const tempFilePath4 = Deno.makeTempFileSync();
const tempDirPath3 = Deno.makeTempDirSync();
console.log("Temp file path 4:", tempFilePath4);
console.log("Temp dir path 3:", tempDirPath3);
```

4.20 timers.ts

```
/**
 * @title Timeouts & Intervals
 * @difficulty beginner
 * @tags cli, deploy, web
 * @run <url>
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Window/setTimeout} MDN: setTimeout
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Window/setInterval} MDN: setInterval
 *
 * Timers are used to schedule functions to happen at a later time.
 */

// Here we create a timer that will print "Hello, World!" to the
console after
// 1 second (1000 milliseconds).
setTimeout(() => console.log("Hello, World!"), 1000);

// You can also cancel a timer after it has been created.
const timerId = setTimeout(() => console.log("No!"), 1000);
clearTimeout(timerId);

// Intervals can be created to repeat a function at a regular
interval.
setInterval(() => console.log("Hey!"), 1000);

// Intervals can also be cancelled.
const intervalId = setInterval(() => console.log("Nope"), 1000);
clearInterval(intervalId);
```

4.21 uuids.ts

```
/**
 * @title Generating & Validating UUIDs
 * @difficulty beginner
 * @tags cli, deploy, web
 * @run <url>
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/Crypto/randomUUID} MDN: crypto.randomUUID
 * @resource {https://doc.deno.land/https://deno.land/std/uuid/mod.ts}
Doc: std/uuid
```



```
 */

// The most basic way to interact with the user is by alerting them,
and waiting
// for them to acknowledge by pressing [Enter].
alert("Please acknowledge the message.");
console.log("The message has been acknowledged.");

// Instead of just an acknowledgement, we can also ask the user for a
yes/no
// response.
const shouldProceed = confirm("Do you want to proceed?");
console.log("Should proceed?", shouldProceed);

// We can also prompt the user for some text input. If the user
cancels the
// prompt, the returned value will be `null`.
const name = prompt("Please enter your name:");
console.log("Name:", name);

// When prompting you can also specify a default value to use if the
user
// cancels the prompt.
const age = prompt("Please enter your age:", "18");
console.log("Age:", age);
```

4.18 reading-files.ts

```
/**
 * @title Reading Files
 * @difficulty beginner
 * @tags cli, deploy
 * @run --allow-read <url>
 * @resource {https://doc.deno.land/deno/stable/~/Deno.readFile} Doc:
Deno.readFile
 * @resource {https://doc.deno.land/deno/stable/~/Deno.open} Doc:
Deno.open
 * @resource {https://doc.deno.land/deno/stable/~/Deno.File} Doc:
Deno.File
 *
 * Many applications need to read files from disk. Deno provides a
simple
 * interface for reading files.
 */

// The easiest way to read a file is to just read the entire contents
into
// memory as bytes.
// deno-lint-ignore no-unused-vars
```

```ts
const bytes = await Deno.readFile("hello.txt");

// Instead of reading the file as bytes, there is a convenience function to
// read the file as a string.
// deno-lint-ignore no-unused-vars
const text = await Deno.readTextFile("hello.txt");

// Often you need more control over when what parts of the file are read.
// For this you start by opening a file to get a `Deno.File` object.
const file = await Deno.open("hello.txt");

// Read some bytes from the beginning of the file. Allow up to 5 to be read but
// also note how many actually were read.
const buffer = new Uint8Array(5);
const bytesRead = await file.read(buffer);
console.log(`Read ${bytesRead} bytes`);

// You can also seek to a known location in the file and read from there.
const pos = await file.seek(6, Deno.SeekMode.Start);
console.log(`Seeked to position ${pos}`);
const buffer2 = new Uint8Array(2);
const bytesRead2 = await file.read(buffer2);
console.log(`Read ${bytesRead2} bytes`);

// You can use rewind back to the start using seek as well.
file.seek(0, Deno.SeekMode.Start);

// Make sure to close the file when you are done.
file.close();

// Synchronous reading is also supported.
Deno.readFileSync("hello.txt");
Deno.readTextFileSync("hello.txt");
const f = Deno.openSync("hello.txt");
f.seekSync(6, Deno.SeekMode.Start);
const buf = new Uint8Array(5);
f.readSync(buf);
f.close();

// The `--allow-read` permission is required to read files.
```

4.19 temporary-files.ts

```ts
/**
 * @title Temporary Files & Directories
 * @difficulty beginner
 * @tags cli
 * @run --allow-read --allow-write <url>
 * @resource {https://doc.deno.land/deno/stable/~/Deno.makeTempFile}
Doc: Deno.makeTempFile
 * @resource {https://doc.deno.land/deno/stable/~/Deno.makeTempDir}
Doc: Deno.makeTempDir
 *
 * Temporary files and directories are used to store data that is not intended
 * to be permanent. For example, as a local cache of downloaded data.
 */

// The `Deno.makeTempFile()` function creates a temporary file in the default
// temporary directory and returns the path to the file.
const tempFilePath = await Deno.makeTempFile();
console.log("Temp file path:", tempFilePath);
await Deno.writeTextFile(tempFilePath, "Hello world!");
const data = await Deno.readTextFile(tempFilePath);
console.log("Temp file data:", data);

// A custom prefix and suffix for the temporary file can be specified.
const tempFilePath2 = await Deno.makeTempFile({
  prefix: "logs_",
  suffix: ".txt",
});
console.log("Temp file path 2:", tempFilePath2);

// The directory that temporary files are created in can be customized too.
// Here we use a relative ./tmp directory.
await Deno.mkdir("./tmp", { recursive: true });
const tempFilePath3 = await Deno.makeTempFile({
  dir: "./tmp",
});
console.log("Temp file path 3:", tempFilePath3);

// A temporary directory can also be created.
const tempDirPath = await Deno.makeTempDir();
console.log("Temp dir path:", tempDirPath);

// It has the same prefix, suffix, and directory options as `makeTempFile()`.
const tempDirPath2 = await Deno.makeTempDir({
  prefix: "logs_",
  suffix: "_folder",
  dir: "./tmp",
});
```

```
    "./pages/index.tsx": $3,
  },
  baseUrl: import.meta.url,
};

export default routes;
```

5.4 test_deps.ts

```
export {
  assert,
  assertEquals,
  assertThrows,
} from "https://deno.land/std@0.119.0/testing/asserts.ts";
```

5.5 components/Footer.tsx

```
/** @jsx h */

import { h, tw } from "../deps.ts";
import { DenoLogo } from "./Logo.tsx";

const FOOTER_LINKS = [
  ["https://deno.land/manual", "Manual"],
  ["https://doc.deno.land/builtin/stable", "Runtime API"],
  ["https://deno.land/std", "Standard Library"],
  ["https://deno.land/x", "Third Party Modules"],
  ["https://deno.com/blog", "Blog"],
  ["https://deno.com/company", "Company"],
];

export function Footer() {
  return (
    <footer class={tw`flex justify-between items-end p-8 pt-32`}>
      <div class={tw`flex align-center`}>
        <DenoLogo />
        <p class={tw`ml-4 font-bold text-xl`}>Deno</p>
      </div>
      <div class={tw`flex flex-col lg:flex-row gap-x-8 gap-y-6 text-
right`}>
        {FOOTER_LINKS.map(([href, text]) => (
          <a href={href} class={tw`text-gray-500
hover:underline`}>{text}</a>
        ))}
      </div>
    </footer>
  );
}
```



```
 *
 * UUIDs (universally unique identifier) can be used to uniquely
identify some
 * object or data.
 */

// A random UUID can be generated using the builtin Web Cryptography
API. This
// type of UUID is also known as UUID v4.
const myUUID = crypto.randomUUID();
console.log("Random UUID:", myUUID);

// The standard library contains some more functions for working with
UUIDs.
import * as uuid from "https://deno.land/std@0.119.0/uuid/mod.ts";

// You can validate that a given string is a valid UUID.
console.log(uuid.validate("not a UUID")); // false
console.log(uuid.validate("6ec0bd7f-11c0-43da-975e-2a8ad9ebae0b")); //
true

// You can also generate a time-based (v1) UUID. By default this uses
system
// time as the time source.
console.log(uuid.v1.generate());

// SHA-1 namespaced (v5) UUIDs can also be generated. For this you
need
// to specify a namespace and data:
const NAMESPACE_URL = "6ba7b810-9dad-11d1-80b4-00c04fd430c8";
const data = new TextEncoder().encode("deno.land");
console.log(await uuid.v5.generate(NAMESPACE_URL, data));
```

4.22 writing-files.ts

```
/**
 * @title Writing Files
 * @difficulty beginner
 * @tags cli
 * @run --allow-read --allow-write <url>
 * @resource {https://doc.deno.land/deno/stable/~/Deno.writeFile} Doc:
Deno.writeFile
 * @resource {https://doc.deno.land/deno/stable/~/Deno.create} Doc:
Deno.create
 * @resource {https://doc.deno.land/deno/stable/~/Deno.File} Doc:
Deno.File
 * @resource {https://developer.mozilla.org/en-
US/docs/Web/API/TextEncoder} MDN: TextEncoder
 *
```

```
 * Many applications need to write files to disk. Deno provides a
simple
 * interface for writing files.
 */

// The easiest way to write a file, is to dump an entire buffer into
the file at
// once.
const bytes = new Uint8Array([72, 101, 108, 108, 111]);
await Deno.writeFile("hello.txt", bytes, { mode: 0o644 });

// You can also write a string instead of a byte array.
await Deno.writeTextFile("hello.txt", "Hello World");

// For more granular writes, open a new file for writing.
const file = await Deno.create("hello.txt");

// You can write chunks of data to the file.
const written = await file.write(bytes);
console.log(`${written} bytes written.`);

// A `file.write` returns the number of bytes written, as it might not
write all
// bytes passed. Use the `writeAll` utility from `std/streams` to make
sure the
// entire buffer is written.
import { writeAll } from
"https://deno.land/std/streams/conversion.ts";
await writeAll(file, new TextEncoder().encode("World!"));

// Make sure to close the file after you are done with it.
file.close();
// Synchronous writing is also supported.
Deno.writeFileSync("hello.txt", bytes);
Deno.writeTextFileSync("hello.txt", "Hello World");
const f = Deno.createSync("hello.txt");
import { writeAllSync } from
"https://deno.land/std/streams/conversion.ts";
writeAllSync(f, new TextEncoder().encode("World!"));
f.close();

// The `--allow-write` permission is required to write files.
```

5 www/

5.1 deps.ts

```
export * from
"https://raw.githubusercontent.com/lucacasonato/fresh/aa1e1bcdb4693610
feb3eb4ad7bba460c54cacaf/runtime.ts";
```



```
export * from
"https://raw.githubusercontent.com/lucacasonato/fresh/aa1e1bcdb4693610
feb3eb4ad7bba460c54cacaf/server.ts";

// x/gfm
export * as gfm from "https://deno.land/x/gfm@0.1.15/mod.ts";

// npm:prismjs
export { default as Prism } from
"https://esm.sh/prismjs@1.25.0?pin=v55";
import "https://esm.sh/prismjs@1.25.0/components/prism-jsx.js?no-
check&pin=v55";
import "https://esm.sh/prismjs@1.25.0/components/prism-
typescript.js?no-check&pin=v55";
import "https://esm.sh/prismjs@1.25.0/components/prism-tsx.js?no-
check&pin=v55";

// npm:twind
export { setup, tw } from "https://esm.sh/twind@0.16.16?pin=v57";
export { virtualSheet } from
"https://esm.sh/twind@0.16.16/sheets?pin=v57";
```

5.2 main.ts

```
/// <reference no-default-lib="true" />
/// <reference lib="dom" />
/// <reference lib="dom.asynciterable" />
/// <reference lib="deno.ns" />
/// <reference lib="deno.unstable" />

import { start } from "./deps.ts";
import routes from "./routes.gen.ts";

await start(routes);
```

5.3 routes.gen.ts

```
// DO NOT EDIT. This file is generated by `fresh`.
// This file SHOULD be checked into source version control.
// To update this file, run `fresh routes`.

import * as $0 from "./pages/_render.ts";
import * as $1 from "./pages/[id].tsx";
import * as $2 from "./pages/gfm.css.ts";
import * as $3 from "./pages/index.tsx";

const routes = {
  pages: {
    "./pages/_render.ts": $0,
    "./pages/[id].tsx": $1,
    "./pages/gfm.css.ts": $2,
```

```
45.09V35.7A10.4 10.4 0 01132.06 25V10.81h4.48v14.13c0 1.48.32 2.76.96
3.85a7.04 7.04 0 006 3.32 7.11 7.11 0 006.07-3.32 7.17 7.17 0 001-
3.85V10.81H155v14.18c0 1.96-.4 3.69-1.2 5.2a9.82 9.82 0 01-3.27 3.7
11.73 11.73 0 01-4.76 1.82v9.38h-4.43z"
      >
      </path>
    </svg>
  );
}

export function CircleArrow(props: { right?: boolean }) {
  return (
    <svg
      width="1em"
      height="1em"
      viewBox="0 0 45.035156 44.982422"
      style={{
        flexShrink: 0,
        transform: props.right ? "rotate(180deg)" : undefined,
      }}
      xmlns="http://www.w3.org/2000/svg"
    >
      <path
        d="M22.517578 44.982422 C19.423813 44.982422 16.505873
44.384772 13.763672 43.189453 C11.021471 42.029291 8.630870 40.429698
6.591797 38.390625 C4.552724 36.351552 2.953131 33.960951 1.792969
31.218750 C0.597650 28.511705 0.000000 25.593766 0.000000 22.464844
C0.000000 19.371078 0.597650 16.453139 1.792969 13.710938 C2.953131
11.003893 4.552724 8.630870 6.591797 6.591797 C8.630870 4.552724
11.021471 2.935553 13.763672 1.740234 C16.505873 0.580072 19.423813 -
0.000000 22.517578 -0.000000 C25.611344 -0.000000 28.529283 0.580072
31.271484 1.740234 C34.013686 2.935553 36.404287 4.552724 38.443359
6.591797 C40.482432 8.630870 42.082025 11.003893 43.242188 13.710938
C44.437506 16.453139 45.035156 19.371078 45.035156 22.464844
C45.035156 25.593766 44.437506 28.511705 43.242188 31.218750
C42.082025 33.960951 40.482432 36.351552 38.443359 38.390625
C36.404287 40.429698 34.013686 42.029291 31.271484 43.189453
C28.529283 44.384772 25.611344 44.982422 22.517578 44.982422 Z
M22.517578 40.500000 C25.013684 40.500000 27.351552 40.025395
29.531250 39.076172 C31.710948 38.126948 33.618156 36.834969 35.252930
35.200195 C36.887703 33.565422 38.179683 31.658214 39.128906 29.478516
C40.042973 27.298817 40.500000 24.960950 40.500000 22.464844
C40.500000 20.003894 40.042973 17.666027 39.128906 15.451172
C38.179683 13.271473 36.887703 11.373055 35.252930 9.755859 C33.618156
8.138664 31.710948 6.855473 29.531250 5.906250 C27.351552 4.957027
25.013684 4.482422 22.517578 4.482422 C20.021472 4.482422 17.683605
4.957027 15.503906 5.906250 C13.324208 6.855473 11.417000 8.138664
9.782227 9.755859 C8.147453 11.373055 6.855473 13.271473 5.906250
15.451172 C4.992183 17.666027 4.535156 20.003894 4.535156 22.464844
```


5.6 components/Header.tsx

```
/** @jsx h */

import { h, tw } from "../deps.ts";
import { DenoLogo } from "./Logo.tsx";

export function Header(props: { noSubtitle?: boolean }) {
  return (
    <header
      class={tw
        `px(3 lg:14) h(12 lg:20) text-gray-500 flex justify-between
items-center`}
    >
      <a
        class={tw`flex items-center flex-shrink-0`}
        href="/"
      >
        <DenoLogo />
        <span
          class={tw
            `ml-4 flex items-baseline gap-x-1 flex-col sm:flex-row
tracking-tighter`}
        >
          <span class={tw`text(2xl gray-900) font-bold leading-none`}>
            Deno
          </span>
          {!props.noSubtitle &&
            (
              <span
                class={tw
                  `font-medium italic text(sm sm:base gray-600) leading-
none`}
              >
                by example
              </span>
            )}
        </span>
      </a>
      <div class={tw`flex items-center gap-6`}>
        <a
          href="https://deno.land/manual"
          class={tw`hover:underline focus:underline`}
        >
          Manual
        </a>
        <a
          href="https://doc.deno.land/builtin/stable"
          class={tw`hover:underline focus:underline`}
```

```
      >
        API
      </a>
    </div>
  </header>
 );
}
```

5.7 components/Logo.tsx

```
/** @jsx h */

import { h } from "../deps.ts";

export function DenoLogo() {
 return (
  <svg
   xmlns="http://www.w3.org/2000/svg"
   width="28"
   height="28"
   viewBox="0 0 512 512"
  >
   <title>Deno logo</title>
   <mask id="a">
    <circle fill="white" cx="256" cy="256" r="230"></circle>
   </mask>
   <circle cx="256" cy="256" r="256"></circle>
   <path
    mask="url(#a)"
    stroke="white"
    stroke-width="25"
    stroke-linecap="round"
    d="M71 319l17-63M107.964 161.095l17-63M36.93 221l17-63M125.964
385l17-63M160.372 486.829l17-63M230 456.329l17-63M206.257 92.587l17-
63M326.395 173.004l17-63M452.182 304.693l17-63M409.124 221l17-
63M299.027 54.558l17-63M400.624 86.058l17-63"
   >
   </path>
   <path
    mask="url(#a)"
    fill="white"
    stroke="black"
    stroke-width="12"
    d="M252.225 344.418c-86.65 2.61-144.576-34.5-144.576-94.363 0-
61.494 60.33-111.145 138.351-111.145 37.683 0 69.532 10.65 94.392
30.092 21.882 17.113 37.521 40.526 45.519 66.312 2.574 8.301 22.863
83.767 61.112 227.295l1.295 4.86-159.793 74.443-1.101-8.063c-8.85-
64.778-16.546-113.338-23.076-145.634-3.237-16.004-6.178-27.96-8.79-
35.794-1.227-3.682-2.355-6.361-3.303-7.952a12.56 12.56 0 00-.03-.05z"
   >
```



```
   </path>
   <circle mask="url(#a)" cx="262" cy="203" r="16"></circle>
  </svg>
 );
}

export function DeployLogo() {
 return (
  <svg
   width="3.37em"
   height="1em"
   viewBox="0 0 155 46"
   xmlns="http://www.w3.org/2000/svg"
  >
   <path
    fill="currentColor"
    d="M12.97 35.9c2.38 0 4.5-.5 6.4-1.53a11.2 11.2 0 004.47-4.43
13.88 13.88 0 001.68-7.02V.22H21.1v14.96H21a9.04 9.04 0 00-3.55-3.56
10 10 0 00-5.2-1.39c-2.24 0-4.28.51-6.1 1.54a11.24 11.24 0 00-4.28
4.37 13.53 13.53 0 00-1.54 6.64c0 2.7.53 5.03 1.59 7.02a11.44 11.44 0
004.47 4.52 13.44 13.44 0 006.58 1.58zm0-3.8c-1.5 0-2.88-.36-4.13-
1.1a8.12 8.12 0 01-2.93-3.12 9.87 9.87 0 01-1.11-4.81c0-1.8.35-3.37
1.06-4.71a8.5 8.5 0 012.93-3.18 7.74 7.74 0 014.14-1.15 7.74 7.74 0
017.02 4.28 9.87 9.87 0 011.1 4.8c0 1.83-.37 3.42-1.1 4.77a7.74 7.74 0
01-6.97 4.23zM44.84 35.33c-2.73 0-5.13-.5-7.22-1.5a12.12 12.12 0 01-
4.85-4.27 12.05 12.05 0 01-1.73-6.5c0-2.43.5-4.6 1.49-6.53.99-1.96
2.37-3.5 4.13-4.62a11.21 11.21 0 016.16-1.68c2.53 0 4.63.54 6.3
1.63a10 10 0 013.8 4.43 14.66 14.66 0 011.24 6.15c0 .77-.04 1.48-.14
2.12H35.75c.16 1.5.66 2.79 1.49 3.84a8.09 8.09 0 003.17 2.36c1.32.51
2.76.77 4.33.77h6.4v3.8h-6.3zm-9.14-13.9h14c0-1.06-.22-2.16-.63-
3.32a6.6 6.6 0 00-2.12-2.93c-1-.83-2.37-1.25-4.13-1.25-1.48 0-2.74.39-
3.8 1.16a7.65 7.65 0 00-2.4 2.83 9.05 9.05 0 00-.92 3.51zM72.2 10.23c-
2.36 0-4.5.51-6.39 1.54a11.43 11.43 0 00-4.52 4.47 13.93 13.93 0 00-
1.63 6.97v22.7h4.42V30.95h.1a9.37 9.37 0 003.56 3.6c1.57.9 3.3 1.35
5.19 1.35a12 12 0 006.06-1.53 10.9 10.9 0 004.28-4.33 13.47 13.47 0
001.58-6.68c0-2.7-.54-5.02-1.63-6.98a10.97 10.97 0 00-4.47-4.56 13.17
13.17 0 00-6.54-1.59zm0 3.8a8 8 0 017.07 4.23 9.87 9.87 0 011.11 4.8c0
1.8-.37 3.37-1.1 4.72a8 8 0 01-2.94 3.17 7.5 7.5 0 01-4.08 1.16c-1.51
0-2.89-.37-4.14-1.11a8.12 8.12 0 01-2.93-3.12c-.7-1.38-1.06-3-1.06-
4.86 0-1.83.35-3.41 1.06-4.76a7.74 7.74 0 017.02-4.23zM90.51
35.33V.23h4.42v35.1h-4.42zM113.5 35.9c-2.5 0-4.72-.56-6.68-1.68a12.86
12.86 0 01-4.56-4.61 13.2 13.2 0 01-1.64-6.54c0-2.4.55-4.57 1.64-
6.5a12.37 12.37 0 014.56-4.61 12.9 12.9 0 016.69-1.73 12.65 12.65 0
0112.88 12.84c0 2.4-.56 4.58-1.68 6.54a12.37 12.37 0 01-11.2 6.3zm0-
3.8a8.02 8.02 0 007.36-4.42 9.99 9.99 0 001.06-4.61 9.7 9.7 0 00-1.06-
4.57 8.28 8.28 0 00-2.93-3.27 8.02 8.02 0 00-4.42-1.2c-1.67 0-3.14.4-
4.43 1.2a8.28 8.28 0 00-2.93 3.27 9.7 9.7 0 00-1.06 4.57c0 1.7.36 3.23
1.06 4.61a8.36 8.36 0 002.93 3.22c1.29.8 2.76 1.2 4.43 1.2zM141.34
```

```
          class={tw`underline`}
        >
          License
        </a>{" "}
        | Inspired by{" "}
        <a href="https://gobyexample.com/" class={tw`underline`}>
          Go by Example
        </a>
      </p>
    </main>
  </Page>
 );
}

function fetcher() {
 return Promise.all(
  TOC.map((id) =>
   Deno.readTextFile(`./data/${id}.ts`)
    .then((text) => parseExample(id, text))
  ),
 );
}
```



```
C4.535156 24.960950 4.992183 27.298817 5.906250 29.478516 C6.855473
31.658214 8.147453 33.565422 9.782227 35.200195 C11.417000 36.834969
13.324208 38.126948 15.503906 39.076172 C17.683605 40.025395 20.021472
40.500000 22.517578 40.500000 Z M18.931641 20.250000 L31.535156
20.250000 C32.132815 20.250000 32.651365 20.469724 33.090820 20.909180
C33.530276 21.348635 33.750000 21.867185 33.750000 22.464844
C33.750000 23.097659 33.530276 23.633787 33.090820 24.073242
C32.651365 24.512698 32.132815 24.732422 31.535156 24.732422
L18.931641 24.732422 L24.679688 30.427734 C24.855470 30.638673
25.004882 30.875975 25.127930 31.139648 C25.250977 31.403322 25.312500
31.693358 25.312500 32.009766 C25.312500 32.642581 25.092776 33.178709
24.653320 33.618164 C24.213865 34.057619 23.677738 34.277344 23.044922
34.277344 C22.728514 34.277344 22.438478 34.215821 22.174805 34.092773
C21.911131 33.969726 21.673829 33.820313 21.462891 33.644531
L11.917969 24.046875 C11.707030 23.871093 11.548829 23.642580
11.443359 23.361328 C11.337890 23.080077 11.285156 22.781252 11.285156
22.464844 C11.285156 22.148436 11.337890 21.858400 11.443359 21.594727
C11.548829 21.331053 11.707030 21.093751 11.917969 20.882812
L21.462891 11.337891 C21.673829 11.162108 21.911131 11.012696
22.174805 10.889648 C22.438478 10.766601 22.728514 10.705078 23.044922
10.705078 C23.677738 10.705078 24.213865 10.924802 24.653320 11.364258
C25.092776 11.803713 25.312500 12.339841 25.312500 12.972656
C25.312500 13.253908 25.250977 13.535155 25.127930 13.816406
C25.004882 14.097658 24.855470 14.326171 24.679688 14.501953 Z
M49.517578 44.982422"
       fill="currentColor"
     >
     </path>
    </svg>
  );
}
```

5.8 components/Page.tsx

```
/** @jsx h */

import { ComponentChildren, h, Head, tw } from "../deps.ts";
import { Footer } from "./Footer.tsx";
import { Header } from "./Header.tsx";

export function Page(props: {
  title: string;
  noSubtitle?: boolean;
  children: ComponentChildren;
}) {
  return (
    <div
      class={tw`min-h-screen grid grid-cols-1`}
      style={"grid-template-rows: auto 1fr auto;"}
```

```
    >
      <Head>
        <link
          rel="shortcut icon"
          href="/favicon.ico"
          type="image/x-icon"
        />
        <title>{props.title}</title>
      </Head>
      <Header noSubtitle={props.noSubtitle} />
      <div>
        {props.children}
      </div>
      <Footer />
    </div>
  );
}
```

5.9 pages/gfm.css.ts

```
import { gfm, HandlerContext } from "../deps.ts";

export const handler = (_ctx: HandlerContext) => {
  return new Response(gfm.CSS, {
    status: 200,
    headers: {
      "content-type": "text/css; charset: utf-8",
    },
  });
};
```

5.10 pages/index.tsx

```
/** @jsx h */
/** @jsxFrag Fragment */
import { TOC } from "../../toc.js";
import { Page } from "../components/Page.tsx";
import { h, Head, PageProps, tw, useData } from "../deps.ts";
import { parseExample } from "../utils/example.ts";

export default function Example(props: PageProps) {
  const examples = useData("", fetcher);

  return (
    <Page title={`Deno by Example`} noSubtitle>
      <Head>
        <meta
          name="description"
          content="Deno by example is a collection of annotated examples
for how to use Deno, and the various features it provides."
        />
```



```
      </Head>
      <main class={tw`max-w-screen-sm mx-auto p-4`}>
        <h1>
          <span class={tw`text(5xl gray-900) tracking-tight font-bold`}>
            Deno
          </span>
          <span
            class={tw
              `text(2xl gray-700) tracking-tight italic font-medium ml-
2`}
          >
            by example
          </span>
        </h1>
        <p class={tw`mt-8 text-gray-900`}>
          Deno is a simple, modern and secure runtime for JavaScript and
          TypeScript that uses V8 and is built in Rust.
        </p>
        <p class={tw`mt-6 text-gray-900`}>
          <i class={tw`italic`}>Deno by example</i>{" "}
          is a collection of annotated examples for how to use Deno, and
the
          various features it provides. It acts as a reference for how
to do
          various things in Deno, but can also be used as a guide to
learn about
          many of the features Deno provides.
        </p>
        <ul class={tw`mt-6 text-gray-900`}>
          {examples.map((example) => (
            <li>
              <a href={`/${example.id}`} class={tw`underline`}>
                {example.title}
              </a>
            </li>
          ))}
        </ul>
        <p class={tw`mt-12 text-gray-500`}>
          <a
            href="https://github.com/denoland/denobyexample"
            class={tw`underline`}
          >
            Source
          </a>{" "}
          |{" "}
          <a

href="https://github.com/denoland/denobyexample/blob/main/LICENSE"
```

```
      </div>

      <div class={tw`col-span-2 mt-12 flex justify-between h-14`}>
        {prev
          ? (
            <a
              href={`/${prev.id}`}
              class={tw
                `w-6/12 text-gray-600 flex items-center gap-3 lg:gap-
2 :hover:text-gray-900`}
            >
              <CircleArrow />
              {prev.title}
            </a>
          )
          : <div class={tw`w-6/12`} />}
        {next && (
          <a
            href={`/${next.id}`}
            class={tw
              `w-6/12 text-gray-600 text-right flex items-center
justify-end gap-3 lg:gap-2 :hover:text-gray-900`}
          >
            {next.title}
            <CircleArrow right />
          </a>
        )}
      </div>
    </main>
  </Page>
);
}

function SnippetComponent(props: {
  filename: string;
  firstOfFile: boolean;
  lastOfFile: boolean;
  snippet: ExampleSnippet;
}) {
  const renderedSnippet = Prism.highlight(
    props.snippet.code,
    Prism.languages.ts,
    "ts",
  );

  return (
    <div
      class={tw
        `grid grid-cols-1 sm:grid-cols-5 gap-x-6  transition duration-
```


5.11 pages/[id].tsx

```
/** @jsx h */
/** @jsxFrag Fragment */
import { Page } from "../components/Page.tsx";
import { CircleArrow, DeployLogo } from "../components/Logo.tsx";
import {
  Fragment,
  h,
  HandlerContext,
  Head,
  PageProps,
  Prism,
  tw,
  useData,
} from "../deps.ts";
import { TOC } from "../../toc.js";
import { DIFFICULTIES, TAGS } from "../utils/constants.ts";
import { ExampleSnippet, parseExample } from "../utils/example.ts";

export default function Example(props: PageProps) {
  const [example, prev, next] = useData(props.params.id as string,
fetcher) ||
    [];
  if (!example) {
    return <div>404 Example Not Found</div>;
  }

  const url = `${props.url.origin}${props.url.pathname}.ts`;

  const description = (example.description || example.title) +
    " -- Deno by example is a collection of annotated examples for how
to use Deno, and the various features it provides.";

  return (
    <Page title={`${example.title} - Deno by Example`}>
      <Head>
        <link rel="stylesheet" href="/gfm.css" />
        <meta name="description" content={description} />
      </Head>
      <main class={tw`max-w-screen-lg mx-auto p-4`}>
        <div class={tw`flex gap-2`}>
          <p
            class={tw`text-gray-500 italic`}
            title={DIFFICULTIES[example.difficulty].description}
          >
            {DIFFICULTIES[example.difficulty].title}
          </p>
          <div class={tw`flex gap-2 items-center`}>
```

```
        {example.tags.map((tag) => (
          <span
            class={tw`text-xs bg-gray-200 py-0.5 px-2 rounded-md`}
            title={TAGS[tag].description}
          >
            {TAGS[tag].title}
          </span>
        ))}
      </div>
    </div>
    <h1 class={tw`mt-2 text-3xl font-bold`}>{example.title}</h1>
    {example.description && (
      <div class={tw`mt-1`}>
        <p class={tw`text-gray-500`}>
          {example.description}
        </p>
      </div>
    )}
    {example.files.map((file) => (
      <div class={tw`mt-10`}>
        {file.snippets.map((snippet, i) => (
          <SnippetComponent
            key={i}
            firstOfFile={i === 0}
            lastOfFile={i === file.snippets.length - 1}
            filename={file.name}
            snippet={snippet}
          />
        ))}
      </div>
    ))}
    <div class={tw`grid grid-cols-1 sm:grid-cols-5 gap-x-6`}>
      <div class={tw`col-span-2 mt-8`} />
      <div class={tw`col-span-3 mt-8`}>
        {example.run && (
          <>
            <p class={tw`text-gray-700`}>
              Run{" "}
              <a href={url} class={tw`hover:underline
focus:underline`}>
                this example
              </a>{" "}
              locally using the Deno CLI:
            </p>
            <pre
              class={tw
                `mt-2 bg-gray-100 p-4 overflow-x-auto text-sm select-
all rounded-md`}
            >
```



```
              deno run {example.run.replace("<url>", url)}
            </pre>
          </>
        )}
        {example.playground && (
          <div class={tw`col-span-3 mt-8`}>
            <p class={tw`text-gray-700`}>
              Try this example in a Deno Deploy playground:
            </p>
            <p class={tw`mt-3`}>
              <a
                class={tw
                  `py-2 px-4 bg-black inline-block text-white text-
base rounded-md opacity-90 hover:opacity-100`}
                href={example.playground}
                target="_blank"
                rel="noreferrer"
                title="Deploy"
              >
                <DeployLogo />
              </a>
            </p>
          </div>
        )}
        {example.additionalResources.length > 0 && (
          <div class={tw`col-span-3 mt-12 pt-6 border-t-1 border-
gray-200`}>
            <p class={tw`text-gray-500`}>
              Additional resources:
            </p>
            <ul class={tw`list-disc list-inside mt-1`}>
              {example.additionalResources.map(([link, title]) => (
                <li
                  class={tw`text-gray-700 hover:text-gray-900`}
                  key={link + title}
                >
                  <a
                    class={tw`hover:underline focus:underline`}
                    href={link}
                  >
                    {title}
                  </a>
                </li>
              ))}
            </ul>
          </div>
        )}
      </div>
```

```
  id: string;
  title: string;
  description: string;
  difficulty: keyof typeof DIFFICULTIES;
  tags: (keyof typeof TAGS)[];
  additionalResources: [string, string][];
  run?: string;
  playground?: string;
  files: ExampleFile[];
}

export interface ExampleFile {
  name: string;
  snippets: ExampleSnippet[];
}

export interface ExampleSnippet {
  text: string;
  code: string;
}

export function parseExample(id: string, file: string): Example {
  // Extract the multi line JS doc comment at the top of the file
  const [, jsdoc, rest] = file.match(/^\s*\/\*\*(.*?)\*\/\s*(.*)/s) ||
[];

  // Extract the @key value pairs from the JS doc comment
  let description = "";
  const kvs: Record<string, string> = {};
  const resources = [];
  for (let line of jsdoc.split("\n")) {
    line = line.trim().replace(/^\*/, "").trim();
    const [, key, value] = line.match(/^\s*@(\w+)\s+(.*)/) || [];
    if (key) {
      if (key === "resource") {
        resources.push(value);
      } else {
        kvs[key] = value.trim();
      }
    } else {
      description += " " + line;
    }
  }
  description = description.trim();

  // Seperate the code into snippets.
  const files: ExampleFile[] = [{
    name: "",
    snippets: [],
```



```
150 ease-in`}
    >
      <div class={tw`py-4 text-gray-700 select-none col-span-2`}>
        {props.snippet.text}
      </div>
      <div
        class={tw`col-span-3 relative bg-gray-100 ${
          props.firstOfFile ? "rounded-t-md" : ""
        } ${props.lastOfFile ? "rounded-b-md" : ""} ${
          props.snippet.code.length === 0 ? "hidden sm:block" : ""
        }`}
      >
        {props.filename && (
          <span
            class={tw
              `font-mono text-xs absolute -top-3 left-4 bg-gray-200 z-10
p-1 rounded-sm ${
                props.firstOfFile ? "block" : "block sm:hidden"
              }`}
          >
            {props.filename}
          </span>
        )}
        <div
          class={tw
            `px-4 py-4 text-sm overflow-scroll sm:overflow-hidden
relative` +
            " highlight"}
        >
          <pre dangerouslySetInnerHTML={{ __html: renderedSnippet }} />
        </div>
      </div>
    </div>
  );
}

async function fetcher(id: string) {
  try {
    const cur = TOC.indexOf(id);
    const prev = TOC[cur - 1];
    const next = TOC[cur + 1];
    const [data, prevData, nextData] = await Promise.all(
      [id, prev, next].map((name) =>
        name ? Deno.readTextFile(`./data/${name}.ts`) :
Promise.resolve("")
      ),
    );
    return [
      parseExample(id, data),
```

```
      prev ? parseExample(prev, prevData) : null,
      next ? parseExample(next, nextData) : null,
    ];
  } catch (err) {
    if (err instanceof Deno.errors.NotFound) {
      return null;
    }
    console.error(err);
  }
}

export const handler = async (ctx: HandlerContext) => {
  const id = ctx.match.id;
  if (id.endsWith(".ts")) {
    const accept = ctx.req.headers.get("accept") || "";
    const acceptsHTML = accept.includes("text/html");
    try {
      const data = await Deno.readTextFile(`./data/${id}`);
      const example = parseExample(id, data);
      if (example.files.length > 1) {
        return new Response(
          "Source for multi file examples can not be viewed",
          {
            status: 400,
          },
        );
      }
      const file = example.files[0];
      let code = "";
      for (const snippet of file.snippets) {
        code += snippet.code + "\n";
      }
      return new Response(code, {
        headers: {
          "content-type": acceptsHTML
            ? "text/plain; charset=utf-8"
            : "application/typescript; charset=utf-8",
        },
      });
    } catch (err) {
      if (err instanceof Deno.errors.NotFound) {
        return new Response("404 Example Not Found", { status: 404 });
      }
      console.error(err);
      return new Response("500 Internal Server Error", { status:
500 });
    }
  }
  return ctx.render!();
};
```

5.12 pages/_render.ts

```
// This module adds twind support.

import { RenderContext, RenderFn, setup, virtualSheet } from
"../deps.ts";

const sheet = virtualSheet();
sheet.reset();
setup({ mode: "strict", sheet });

export function render(ctx: RenderContext, render: RenderFn) {
  const snapshot = ctx.state.get("twindSnapshot") as unknown[] | null;
  sheet.reset(snapshot || undefined);
  render();
  ctx.styles.splice(0, ctx.styles.length, ...sheet.target);
  const newSnapshot = sheet.reset();
  ctx.state.set("twindSnapshot", newSnapshot);
}
```

5.13 static/robots.txt

5.14 utils/constants.ts

```
export const TAGS = {
  cli: {
    title: "cli",
    description: "Works in Deno CLI",
  },
  deploy: {
    title: "deploy",
    description: "Works on Deno Deploy",
  },
  web: {
    title: "web",
    description: "Works in on the Web",
  },
};

export const DIFFICULTIES = {
  "beginner": {
    title: "Beginner",
    description: "No significant prior knowledge is required for this
example.",
  },
  "intermediate": {
    title: "Intermediate",
    description: "Some prior knowledge is needed for this example.",
  },
};
```

5.15 utils/example.ts

```
import { DIFFICULTIES, TAGS } from "./constants.ts";

export interface Example {
```

5.16 utils/example_test.ts

```ts
import { assertEquals, assertThrows } from "../test_deps.ts";
import { Example, parseExample } from "./example.ts";

Deno.test("parse jsdoc", () => {
  const example = `
/**
 * @title Input Prompts
 * @difficulty beginner
 * @tags cli, web   ,   deploy
 * @run <url>
 * @playground <url>
 *
 * Prompts are used to ask the user for input or feedback on actions.
 */
`;
  const expected: Example = {
    id: "input-prompts",
    title: "Input Prompts",
    description:
      "Prompts are used to ask the user for input or feedback on
actions.",
    difficulty: "beginner",
    tags: ["cli", "web", "deploy"],
    additionalResources: [],
    run: "<url>",
    playground: "<url>",
    files: [{
      name: "",
      snippets: [],
    }],
  };
  const actual = parseExample("input-prompts", example);
  assertEquals(actual, expected);
});

Deno.test("parse jsdoc unknown tag", () => {
  const example = `
/**
 * @title abc
 * @difficulty beginner
 * @tags foo, cli, deploy
 * @run <url>
 *
 * xyz
 */
`;
  assertThrows(
    () => {
```



```ts
  }];
  let parseMode = "code";
  let currentFile = files[0];
  let text = "";
  let code = "";

  for (const line of rest.split("\n")) {
    const trimmedLine = line.trim();
    if (parseMode == "code") {
      if (line.startsWith("// File:")) {
        if (text || code.trimEnd()) {
          code = code.trimEnd();
          currentFile.snippets.push({ text, code });
          text = "";
          code = "";
        }
        const name = line.slice(8).trim();
        if (currentFile.snippets.length == 0) {
          currentFile.name = name;
        } else {
          currentFile = {
            name,
            snippets: [],
          };
          files.push(currentFile);
        }
      } else if (line.startsWith("/* File:")) {
        if (text || code.trimEnd()) {
          code = code.trimEnd();
          currentFile.snippets.push({ text, code });
          text = "";
          code = "";
        }
        const name = line.slice(8).trim();
        if (currentFile.snippets.length == 0) {
          currentFile.name = name;
        } else {
          currentFile = {
            name,
            snippets: [],
          };
          files.push(currentFile);
        }
        parseMode = "file";
      } else if (
        trimmedLine.startsWith("// deno-lint-ignore") ||
        trimmedLine.startsWith("//deno-lint-ignore")
      ) {
```

```
      // skip lint directives
    } else if (trimmedLine.startsWith("//-")) {
      code += line.replace("//-", "//") + "\n";
    } else if (trimmedLine.startsWith("//")) {
      if (text || code.trimEnd()) {
        code = code.trimEnd();
        currentFile.snippets.push({ text, code });
      }
      text = trimmedLine.slice(2).trim();
      code = "";
      parseMode = "comment";
    } else {
      code += line + "\n";
    }
  } else if (parseMode == "comment") {
    if (
      trimmedLine.startsWith("// deno-lint-ignore") ||
      trimmedLine.startsWith("//deno-lint-ignore")
    ) {
      // skip lint directives
    } else if (trimmedLine.startsWith("//")) {
      text += " " + trimmedLine.slice(2).trim();
    } else {
      code += line + "\n";
      parseMode = "code";
    }
  } else if (parseMode == "file") {
    if (line == "*/") {
      parseMode = "code";
    } else {
      code += line + "\n";
    }
  }
}
if (text || code.trimEnd()) {
  code = code.trimEnd();
  currentFile.snippets.push({ text, code });
}

if (!kvs.title) {
  throw new Error("Missing title in JS doc comment.");
}

const tags = kvs.tags.split(",").map((s) => s.trim() as keyof typeof
TAGS);
for (const tag of tags) {
  if (!TAGS[tag]) {
    throw new Error(`Unknown tag '${tag}'`);
  }
```



```
}

const difficulty = kvs.difficulty as keyof typeof DIFFICULTIES;
if (!DIFFICULTIES[difficulty]) {
  throw new Error(`Unknown difficulty '${difficulty}'`);
}

const additionalResources: [string, string][] = [];
for (const resource of resources) {
  // @resource {https://deno.land/std/http/server.ts}
std/http/server.ts
  const [_, url, title] = resource.match(/^\{(.*?)\}\s(.*)/) || [];
  if (!url || !title) {
    throw new Error(`Invalid resource: ${resource}`);
  }
  additionalResources.push([url, title]);
}

return {
  id,
  title: kvs.title,
  description,
  difficulty,
  tags,
  additionalResources,
  run: kvs.run,
  playground: kvs.playground,
  files,
};
}
```

```
`;
  assertThrows(
    () => {
      parseExample("abc", example);
    },
    Error,
    "Invalid resource",
  );
});

const BASIC_JSDOC = `
/**
 * @title abc
 * @difficulty beginner
 * @tags cli, deploy
 */
`;
const EXPECTED_BASIC: Example = {
  id: "abc",
  title: "abc",
  description: "",
  difficulty: "beginner",
  tags: ["cli", "deploy"],
  additionalResources: [],
  run: undefined,
  playground: undefined,
  files: [{
    name: "",
    snippets: [],
  }],
};

Deno.test("parse snippets", () => {
  const example = `${BASIC_JSDOC}

// snippet 1
code 1;

// snippet 2

  // snippet 3
  foo;

/** still snippet 3 */

// ending snippet
`;
  const expected: Example = {
    ...EXPECTED_BASIC,
```



```
      parseExample("abc", example);
    },
    Error,
    "Unknown tag 'foo'",
  );
});

Deno.test("parse jsdoc unknown difficulty", () => {
  const example = `
/**
 * @title abc
 * @difficulty garbage
 * @tags cli, deploy
 * @run <url>
 *
 * xyz
 */
`;
  assertThrows(
    () => {
      parseExample("abc", example);
    },
    Error,
    "Unknown difficulty 'garbage'",
  );
});

Deno.test("parse jsdoc missing title", () => {
  const example = `
/**
 * @difficulty garbage
 * @tags cli, deploy
 * @run <url>
 *
 * xyz
 */
`;
  assertThrows(
    () => {
      parseExample("abc", example);
    },
    Error,
    "Missing title",
  );
});

Deno.test("parse jsdoc no run", () => {
  const example = `
```

```
/**
 * @title abc
 * @difficulty beginner
 * @tags cli, deploy
 *
 * xyz
 */
`;
  const expected: Example = {
    id: "abc",
    title: "abc",
    description: "xyz",
    difficulty: "beginner",
    tags: ["cli", "deploy"],
    additionalResources: [],
    run: undefined,
    playground: undefined,
    files: [{
      name: "",
      snippets: [],
    }],
  };
  const actual = parseExample("abc", example);
  assertEquals(actual, expected);
});

Deno.test("parse jsdoc no description", () => {
  const example = `
/**
 * @title abc
 * @difficulty beginner
 * @tags cli, deploy
 */
`;
  const expected: Example = {
    id: "abc",
    title: "abc",
    description: "",
    difficulty: "beginner",
    tags: ["cli", "deploy"],
    additionalResources: [],
    run: undefined,
    playground: undefined,
    files: [{
      name: "",
      snippets: [],
    }],
  };
  const actual = parseExample("abc", example);
```



```
  assertEquals(actual, expected);
});

Deno.test("parse jsdoc resources", () => {
  const example = `
/**
 * @title abc
 * @difficulty beginner
 * @tags cli, deploy
 * @resource {https://deno.land#install} Deno: Installation
 * @resource
{https://deno.land/manual/getting_started/setup_your_environment} Deno
Manual: Setup your environemnt
 */
`;
  const expected: Example = {
    id: "abc",
    title: "abc",
    description: "",
    difficulty: "beginner",
    tags: ["cli", "deploy"],
    additionalResources: [
      ["https://deno.land#install", "Deno: Installation"],
      [

"https://deno.land/manual/getting_started/setup_your_environment",
        "Deno Manual: Setup your environemnt",
      ],
    ],
    run: undefined,
    playground: undefined,
    files: [{
      name: "",
      snippets: [],
    }],
  };
  const actual = parseExample("abc", example);
  assertEquals(actual, expected);
});

Deno.test("parse jsdoc resources broken", () => {
  const example = `
/**
 * @title abc
 * @difficulty beginner
 * @tags cli, deploy
 * @resource {}
 */
```

deno official examples

目录


```
    files: [
      {
        name: "",
        snippets: [
          {
            text: "snippet 1",
            code: "code 1;",
          },
          {
            text: "snippet 2",
            code: "",
          },
          {
            text: "snippet 3",
            code: "  foo;\n\n/** still snippet 3 */",
          },
          {
            text: "ending snippet",
            code: "",
          },
        ],
      },
    ],
  };
  const actual = parseExample("abc", example);
  assertEquals(actual, expected);
});

Deno.test("parser ignores deno-lint-ignore", () => {
  const example = `${BASIC_JSDOC}
// deno-lint-ignore no-unused-vars
const foo = "bar";

// comment
// deno-lint-ignore no-unused-vars
const bar = "baz";

// comment 2
foo;
// deno-lint-ignore no-unused-vars
bar;
  // deno-lint-ignore no-unused-vars
`;

  const expected: Example = {
    ...EXPECTED_BASIC,
    files: [
      {
        name: "",
        snippets: [
```

```
        {
          text: "",
          code: `const foo = "bar";`,
        },
        {
          text: "comment",
          code: `const bar = "baz";`,
        },
        {
          text: "comment 2",
          code: `foo;\nbar;`,
        },
      ],
    },
  ],
};
const actual = parseExample("abc", example);
assertEquals(actual, expected);
});

Deno.test("parse multiple files", () => {
  const example = `${BASIC_JSDOC}

// File: main.ts
// Main file
import "./foo.ts";

// File: foo.ts
// Hello
globalThis.Hello = "World";

// Back to main
foo;
  // File: main.ts

// foo
// File: bar.ts

/* File: hey.json
{
  "hello": "world"
}
*/
`;
  const expected: Example = {
    ...EXPECTED_BASIC,
    files: [
      {
        name: "main.ts",
        snippets: [
          {
```



```
            text: "Main file",
            code: `import "./foo.ts";`,
          },
        ],
      },
      {
        name: "foo.ts",
        snippets: [
          {
            text: "Hello",
            code: `globalThis.Hello = "World";`,
          },
          {
            code: "foo;",
            text: "Back to main",
          },
          {
            code: "",
            text: "File: main.ts",
          },
          {
            text: "foo File: bar.ts",
            code: "",
          },
        ],
      },
      {
        name: "hey.json",
        snippets: [
          {
            text: "",
            code: `{
  "hello": "world"
}`,
          },
        ],
      },
    ],
  };
  const actual = parseExample("abc", example);
  assertEquals(actual, expected);
});
```