/\*\*

\* The `repl` module provides a Read-Eval-Print-Loop (REPL) implementation that

\* is available both as a standalone program or includible in other applications.

\* It can be accessed using:

\*

\* ```js

\* const repl = require('repl');

\* ```

\* @see [source](https://github.com/nodejs/node/blob/v17.0.0/lib/repl.js)

\*/

declare module 'repl' {

import { Interface, Completer, AsyncCompleter } from 'node:readline';

import { Context } from 'node:vm';

import { InspectOptions } from 'node:util';

interface ReplOptions {

/\*\*

\* The input prompt to display.

\* @default "> "

\*/

prompt?: string | undefined;

/\*\*

\* The `Readable` stream from which REPL input will be read.

\* @default process.stdin

\*/

input?: NodeJS.ReadableStream | undefined;

/\*\*

\* The `Writable` stream to which REPL output will be written.

\* @default process.stdout

\*/

output?: NodeJS.WritableStream | undefined;

/\*\*

\* If `true`, specifies that the output should be treated as a TTY terminal, and have

\* ANSI/VT100 escape codes written to it.

\* Default: checking the value of the `isTTY` property on the output stream upon

\* instantiation.

\*/

terminal?: boolean | undefined;

/\*\*

\* The function to be used when evaluating each given line of input.

\* Default: an async wrapper for the JavaScript `eval()` function. An `eval` function can

\* error with `repl.Recoverable` to indicate the input was incomplete and prompt for

\* additional lines.

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_default\_evaluation

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_custom\_evaluation\_functions

\*/

eval?: REPLEval | undefined;

/\*\*

\* Defines if the repl prints output previews or not.

\* @default `true` Always `false` in case `terminal` is falsy.

\*/

preview?: boolean | undefined;

/\*\*

\* If `true`, specifies that the default `writer` function should include ANSI color

\* styling to REPL output. If a custom `writer` function is provided then this has no

\* effect.

\* Default: the REPL instance's `terminal` value.

\*/

useColors?: boolean | undefined;

/\*\*

\* If `true`, specifies that the default evaluation function will use the JavaScript

\* `global` as the context as opposed to creating a new separate context for the REPL

\* instance. The node CLI REPL sets this value to `true`.

\* Default: `false`.

\*/

useGlobal?: boolean | undefined;

/\*\*

\* If `true`, specifies that the default writer will not output the return value of a

\* command if it evaluates to `undefined`.

\* Default: `false`.

\*/

ignoreUndefined?: boolean | undefined;

/\*\*

\* The function to invoke to format the output of each command before writing to `output`.

\* Default: a wrapper for `util.inspect`.

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_customizing\_repl\_output

\*/

writer?: REPLWriter | undefined;

/\*\*

\* An optional function used for custom Tab auto completion.

\*

\* @see https://nodejs.org/dist/latest-v11.x/docs/api/readline.html#readline\_use\_of\_the\_completer\_function

\*/

completer?: Completer | AsyncCompleter | undefined;

/\*\*

\* A flag that specifies whether the default evaluator executes all JavaScript commands in

\* strict mode or default (sloppy) mode.

\* Accepted values are:

\* - `repl.REPL\_MODE\_SLOPPY` - evaluates expressions in sloppy mode.

\* - `repl.REPL\_MODE\_STRICT` - evaluates expressions in strict mode. This is equivalent to

\* prefacing every repl statement with `'use strict'`.

\*/

replMode?: typeof REPL\_MODE\_SLOPPY | typeof REPL\_MODE\_STRICT | undefined;

/\*\*

\* Stop evaluating the current piece of code when `SIGINT` is received, i.e. `Ctrl+C` is

\* pressed. This cannot be used together with a custom `eval` function.

\* Default: `false`.

\*/

breakEvalOnSigint?: boolean | undefined;

}

type REPLEval = (this: REPLServer, evalCmd: string, context: Context, file: string, cb: (err: Error | null, result: any) => void) => void;

type REPLWriter = (this: REPLServer, obj: any) => string;

/\*\*

\* This is the default "writer" value, if none is passed in the REPL options,

\* and it can be overridden by custom print functions.

\*/

const writer: REPLWriter & {

options: InspectOptions;

};

type REPLCommandAction = (this: REPLServer, text: string) => void;

interface REPLCommand {

/\*\*

\* Help text to be displayed when `.help` is entered.

\*/

help?: string | undefined;

/\*\*

\* The function to execute, optionally accepting a single string argument.

\*/

action: REPLCommandAction;

}

/\*\*

\* Instances of `repl.REPLServer` are created using the {@link start} method

\* or directly using the JavaScript `new` keyword.

\*

\* ```js

\* const repl = require('repl');

\*

\* const options = { useColors: true };

\*

\* const firstInstance = repl.start(options);

\* const secondInstance = new repl.REPLServer(options);

\* ```

\* @since v0.1.91

\*/

class REPLServer extends Interface {

/\*\*

\* The `vm.Context` provided to the `eval` function to be used for JavaScript

\* evaluation.

\*/

readonly context: Context;

/\*\*

\* @deprecated since v14.3.0 - Use `input` instead.

\*/

readonly inputStream: NodeJS.ReadableStream;

/\*\*

\* @deprecated since v14.3.0 - Use `output` instead.

\*/

readonly outputStream: NodeJS.WritableStream;

/\*\*

\* The `Readable` stream from which REPL input will be read.

\*/

readonly input: NodeJS.ReadableStream;

/\*\*

\* The `Writable` stream to which REPL output will be written.

\*/

readonly output: NodeJS.WritableStream;

/\*\*

\* The commands registered via `replServer.defineCommand()`.

\*/

readonly commands: NodeJS.ReadOnlyDict<REPLCommand>;

/\*\*

\* A value indicating whether the REPL is currently in "editor mode".

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_commands\_and\_special\_keys

\*/

readonly editorMode: boolean;

/\*\*

\* A value indicating whether the `\_` variable has been assigned.

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_assignment\_of\_the\_underscore\_variable

\*/

readonly underscoreAssigned: boolean;

/\*\*

\* The last evaluation result from the REPL (assigned to the `\_` variable inside of the REPL).

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_assignment\_of\_the\_underscore\_variable

\*/

readonly last: any;

/\*\*

\* A value indicating whether the `\_error` variable has been assigned.

\*

\* @since v9.8.0

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_assignment\_of\_the\_underscore\_variable

\*/

readonly underscoreErrAssigned: boolean;

/\*\*

\* The last error raised inside the REPL (assigned to the `\_error` variable inside of the REPL).

\*

\* @since v9.8.0

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_assignment\_of\_the\_underscore\_variable

\*/

readonly lastError: any;

/\*\*

\* Specified in the REPL options, this is the function to be used when evaluating each

\* given line of input. If not specified in the REPL options, this is an async wrapper

\* for the JavaScript `eval()` function.

\*/

readonly eval: REPLEval;

/\*\*

\* Specified in the REPL options, this is a value indicating whether the default

\* `writer` function should include ANSI color styling to REPL output.

\*/

readonly useColors: boolean;

/\*\*

\* Specified in the REPL options, this is a value indicating whether the default `eval`

\* function will use the JavaScript `global` as the context as opposed to creating a new

\* separate context for the REPL instance.

\*/

readonly useGlobal: boolean;

/\*\*

\* Specified in the REPL options, this is a value indicating whether the default `writer`

\* function should output the result of a command if it evaluates to `undefined`.

\*/

readonly ignoreUndefined: boolean;

/\*\*

\* Specified in the REPL options, this is the function to invoke to format the output of

\* each command before writing to `outputStream`. If not specified in the REPL options,

\* this will be a wrapper for `util.inspect`.

\*/

readonly writer: REPLWriter;

/\*\*

\* Specified in the REPL options, this is the function to use for custom Tab auto-completion.

\*/

readonly completer: Completer | AsyncCompleter;

/\*\*

\* Specified in the REPL options, this is a flag that specifies whether the default `eval`

\* function should execute all JavaScript commands in strict mode or default (sloppy) mode.

\* Possible values are:

\* - `repl.REPL\_MODE\_SLOPPY` - evaluates expressions in sloppy mode.

\* - `repl.REPL\_MODE\_STRICT` - evaluates expressions in strict mode. This is equivalent to

\* prefacing every repl statement with `'use strict'`.

\*/

readonly replMode: typeof REPL\_MODE\_SLOPPY | typeof REPL\_MODE\_STRICT;

/\*\*

\* NOTE: According to the documentation:

\*

\* > Instances of `repl.REPLServer` are created using the `repl.start()` method and

\* > \_should not\_ be created directly using the JavaScript `new` keyword.

\*

\* `REPLServer` cannot be subclassed due to implementation specifics in NodeJS.

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_class\_replserver

\*/

private constructor();

/\*\*

\* The `replServer.defineCommand()` method is used to add new `.`\-prefixed commands

\* to the REPL instance. Such commands are invoked by typing a `.` followed by the`keyword`. The `cmd` is either a `Function` or an `Object` with the following

\* properties:

\*

\* The following example shows two new commands added to the REPL instance:

\*

\* ```js

\* const repl = require('repl');

\*

\* const replServer = repl.start({ prompt: '> ' });

\* replServer.defineCommand('sayhello', {

\* help: 'Say hello',

\* action(name) {

\* this.clearBufferedCommand();

\* console.log(`Hello, ${name}!`);

\* this.displayPrompt();

\* }

\* });

\* replServer.defineCommand('saybye', function saybye() {

\* console.log('Goodbye!');

\* this.close();

\* });

\* ```

\*

\* The new commands can then be used from within the REPL instance:

\*

\* ```console

\* > .sayhello Node.js User

\* Hello, Node.js User!

\* > .saybye

\* Goodbye!

\* ```

\* @since v0.3.0

\* @param keyword The command keyword (\*without\* a leading `.` character).

\* @param cmd The function to invoke when the command is processed.

\*/

defineCommand(keyword: string, cmd: REPLCommandAction | REPLCommand): void;

/\*\*

\* The `replServer.displayPrompt()` method readies the REPL instance for input

\* from the user, printing the configured `prompt` to a new line in the `output`and resuming the `input` to accept new input.

\*

\* When multi-line input is being entered, an ellipsis is printed rather than the

\* 'prompt'.

\*

\* When `preserveCursor` is `true`, the cursor placement will not be reset to `0`.

\*

\* The `replServer.displayPrompt` method is primarily intended to be called from

\* within the action function for commands registered using the`replServer.defineCommand()` method.

\* @since v0.1.91

\*/

displayPrompt(preserveCursor?: boolean): void;

/\*\*

\* The `replServer.clearBufferedCommand()` method clears any command that has been

\* buffered but not yet executed. This method is primarily intended to be

\* called from within the action function for commands registered using the`replServer.defineCommand()` method.

\* @since v9.0.0

\*/

clearBufferedCommand(): void;

/\*\*

\* Initializes a history log file for the REPL instance. When executing the

\* Node.js binary and using the command-line REPL, a history file is initialized

\* by default. However, this is not the case when creating a REPL

\* programmatically. Use this method to initialize a history log file when working

\* with REPL instances programmatically.

\* @since v11.10.0

\* @param historyPath the path to the history file

\* @param callback called when history writes are ready or upon error

\*/

setupHistory(path: string, callback: (err: Error | null, repl: this) => void): void;

/\*\*

\* events.EventEmitter

\* 1. close - inherited from `readline.Interface`

\* 2. line - inherited from `readline.Interface`

\* 3. pause - inherited from `readline.Interface`

\* 4. resume - inherited from `readline.Interface`

\* 5. SIGCONT - inherited from `readline.Interface`

\* 6. SIGINT - inherited from `readline.Interface`

\* 7. SIGTSTP - inherited from `readline.Interface`

\* 8. exit

\* 9. reset

\*/

addListener(event: string, listener: (...args: any[]) => void): this;

addListener(event: 'close', listener: () => void): this;

addListener(event: 'line', listener: (input: string) => void): this;

addListener(event: 'pause', listener: () => void): this;

addListener(event: 'resume', listener: () => void): this;

addListener(event: 'SIGCONT', listener: () => void): this;

addListener(event: 'SIGINT', listener: () => void): this;

addListener(event: 'SIGTSTP', listener: () => void): this;

addListener(event: 'exit', listener: () => void): this;

addListener(event: 'reset', listener: (context: Context) => void): this;

emit(event: string | symbol, ...args: any[]): boolean;

emit(event: 'close'): boolean;

emit(event: 'line', input: string): boolean;

emit(event: 'pause'): boolean;

emit(event: 'resume'): boolean;

emit(event: 'SIGCONT'): boolean;

emit(event: 'SIGINT'): boolean;

emit(event: 'SIGTSTP'): boolean;

emit(event: 'exit'): boolean;

emit(event: 'reset', context: Context): boolean;

on(event: string, listener: (...args: any[]) => void): this;

on(event: 'close', listener: () => void): this;

on(event: 'line', listener: (input: string) => void): this;

on(event: 'pause', listener: () => void): this;

on(event: 'resume', listener: () => void): this;

on(event: 'SIGCONT', listener: () => void): this;

on(event: 'SIGINT', listener: () => void): this;

on(event: 'SIGTSTP', listener: () => void): this;

on(event: 'exit', listener: () => void): this;

on(event: 'reset', listener: (context: Context) => void): this;

once(event: string, listener: (...args: any[]) => void): this;

once(event: 'close', listener: () => void): this;

once(event: 'line', listener: (input: string) => void): this;

once(event: 'pause', listener: () => void): this;

once(event: 'resume', listener: () => void): this;

once(event: 'SIGCONT', listener: () => void): this;

once(event: 'SIGINT', listener: () => void): this;

once(event: 'SIGTSTP', listener: () => void): this;

once(event: 'exit', listener: () => void): this;

once(event: 'reset', listener: (context: Context) => void): this;

prependListener(event: string, listener: (...args: any[]) => void): this;

prependListener(event: 'close', listener: () => void): this;

prependListener(event: 'line', listener: (input: string) => void): this;

prependListener(event: 'pause', listener: () => void): this;

prependListener(event: 'resume', listener: () => void): this;

prependListener(event: 'SIGCONT', listener: () => void): this;

prependListener(event: 'SIGINT', listener: () => void): this;

prependListener(event: 'SIGTSTP', listener: () => void): this;

prependListener(event: 'exit', listener: () => void): this;

prependListener(event: 'reset', listener: (context: Context) => void): this;

prependOnceListener(event: string, listener: (...args: any[]) => void): this;

prependOnceListener(event: 'close', listener: () => void): this;

prependOnceListener(event: 'line', listener: (input: string) => void): this;

prependOnceListener(event: 'pause', listener: () => void): this;

prependOnceListener(event: 'resume', listener: () => void): this;

prependOnceListener(event: 'SIGCONT', listener: () => void): this;

prependOnceListener(event: 'SIGINT', listener: () => void): this;

prependOnceListener(event: 'SIGTSTP', listener: () => void): this;

prependOnceListener(event: 'exit', listener: () => void): this;

prependOnceListener(event: 'reset', listener: (context: Context) => void): this;

}

/\*\*

\* A flag passed in the REPL options. Evaluates expressions in sloppy mode.

\*/

const REPL\_MODE\_SLOPPY: unique symbol;

/\*\*

\* A flag passed in the REPL options. Evaluates expressions in strict mode.

\* This is equivalent to prefacing every repl statement with `'use strict'`.

\*/

const REPL\_MODE\_STRICT: unique symbol;

/\*\*

\* The `repl.start()` method creates and starts a {@link REPLServer} instance.

\*

\* If `options` is a string, then it specifies the input prompt:

\*

\* ```js

\* const repl = require('repl');

\*

\* // a Unix style prompt

\* repl.start('$ ');

\* ```

\* @since v0.1.91

\*/

function start(options?: string | ReplOptions): REPLServer;

/\*\*

\* Indicates a recoverable error that a `REPLServer` can use to support multi-line input.

\*

\* @see https://nodejs.org/dist/latest-v10.x/docs/api/repl.html#repl\_recoverable\_errors

\*/

class Recoverable extends SyntaxError {

err: Error;

constructor(err: Error);

}

}

declare module 'node:repl' {

export \* from 'repl';

}