Contents

[(ns sudoku.game) 3](#_Toc154573250)

[print-sudoku 3](#_Toc154573251)

[get-input-difficulty 3](#_Toc154573252)

[get-user-name 3](#_Toc154573253)

[valid-user-input? 3](#_Toc154573254)

[valid-number-entered? 4](#_Toc154573255)

[valid-position? 4](#_Toc154573256)

[sudoku-filled? 4](#_Toc154573257)

[position-empty? 5](#_Toc154573258)

[row-free-of-num? 5](#_Toc154573259)

[column-free-of-num? 5](#_Toc154573260)

[block-free-of-num? 6](#_Toc154573261)

[sudoku-rules-valid? 6](#_Toc154573262)

[get-empty-position-dash 6](#_Toc154573263)

[solver-dash 7](#_Toc154573264)

[move-correct 7](#_Toc154573265)

[(ns sudoku.stopwatch) 8](#_Toc154573266)

[start-stopwatch 8](#_Toc154573267)

[stop-stopwatch 8](#_Toc154573268)

[format-duration 8](#_Toc154573269)

[(ns sudoku.api-generator) 9](#_Toc154573270)

[fetch-sudoku 9](#_Toc154573271)

[remove-positions 9](#_Toc154573272)

[get-new-generated-sudoku 9](#_Toc154573273)

[(ns sudoku.database) 10](#_Toc154573274)

[format-date-to-string 10](#_Toc154573275)

[execute-statements 10](#_Toc154573276)

[save-game 10](#_Toc154573277)

[get-rankings-by-difficulty 11](#_Toc154573278)

[print-rankings-by-difficulty 11](#_Toc154573279)

[custom-order 11](#_Toc154573280)

[group-by-difficulty 12](#_Toc154573281)

[print-all-rankings 12](#_Toc154573282)

# (ns sudoku.game)

## print-sudoku

(**print-sudoku** sudoku initial-sudoku)

Prints a Sudoku grid in the specified format.

Args:

- sudoku (vector of vectors): The newest state of the Sudoku.

- initial-sudoku (vector of vectors): The initial Sudoku provided at the beginning of the game.

Note:

- Initial Sudoku numbers are printed in yellow.

- User-added numbers are printed in white.

## get-input-difficulty

(**get-input-difficulty**)

Prompts the player to enter the difficulty level. Prompt will keep showing up if valid difficulty isn’t entered

Returns:

- String: The selected difficulty level (extreme easy, easy, medium, hard).

## get-user-name

(**get-user-name**)

Takes player input for their name.

Returns:

- String: player’s input

## valid-user-input?

(**valid-user-input?** input)

Expects the player to enter a number, row, and column separated by spaces. Function maps through elements of the input that are separated by space and then it tries to convert them to Integer. These elements are stored in a vector and function returns true if size of the vector is 3 and if all of these elements are integers.

Args:

- input (string): The player's input.

Returns:

- Boolean: True if the input is valid; otherwise, false.

## valid-number-entered?

(**valid-number-entered?** number)

Checks if the number entered by the player for the Sudoku is valid – that number is in the range 1-9

Args:

- number (integer): The number to be checked.

Returns:

- Boolean: True if the number is valid, otherwise, false.

## valid-position?

(**valid-position?** index)

Checks if the index entered by the player for a row or column is valid. Number that player entered will be decremented by one and then checked if it is in a range 0-8.

Args:

- index (integer): The index entered by the player.

Returns:

- Boolean: True if the index is valid, otherwise, false.

## sudoku-filled?

(**sudoku-filled?** sudoku)

Returns true if the Sudoku is completely filled. Sudoku is filled if there are no more "-" in sudoku grid.

Args:

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- Boolean: True if the Sudoku is filled; otherwise, false.

## position-empty?

(**position-empty?** row column sudoku)

Checks if a specific position in the Sudoku grid is empty. Position defined by given row and column index is empty if its value is equal to "-".

Args:

- row (integer): Row index.

- column (integer): Column index.

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- Boolean: True if the position is empty; otherwise, false.

## row-free-of-num?

(**row-free-of-num?** number row sudoku)

Checks if a given number can be placed in a specific row of the Sudoku without violating rules. Function compares number that player wants to enter with all numbers in the given row in the sudoku.

Args:

- number (integer): The number to be checked.

- row (integer): Row index.

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- Boolean: True if the row is free from the given number; otherwise, false.

## column-free-of-num?

(**column-free-of-num?** number column sudoku)

Checks if a given number can be placed in a specific column of the Sudoku without violating rules. This is done by iterating thorough all rows from sudoku and taking nth column from every row, comparing its elements with the given number.

Args:

- number (integer): The number to be checked.

- column (integer): Column index.

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- Boolean: True if the column is free from the given number; otherwise, false.

## block-free-of-num?

(**block-free-of-num?** number row column sudoku)

Checks if a given number can be placed in a specific 3x3 submatrix of the Sudoku without violating rules.

Args:

- number (integer): The number to be checked.

- row (integer): Row index.

- column (integer): Column index.

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- Boolean: True if the submatrix is free from the given number; otherwise, false.

## sudoku-rules-valid?

(**sudoku-rules-valid?** num row col grid)

Checks if placing a given number at a specific position in the Sudoku grid follows rules. This function calls helper function that check each rule.

Args:

- num (integer): The number to be placed.

- row (integer): Row index.

- col (integer): Column index.

- grid (vector of vectors): The Sudoku grid.

Returns:

- Boolean: True if rules are valid; otherwise, false.

## get-empty-position-dash

(**get-empty-position-dash** sudoku)

Returns the indices of the first empty position in the Sudoku grid.

Args:

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- Vector: [row col] representing the indices of the first empty position.

## solver-dash

(**solver-dash** sudoku)

Recursively solves a Sudoku board using a backtracking algorithm.

Args:

- board (vector of vectors): The Sudoku board.

Algorithm:

The function uses a recursive backtracking approach to fill in the empty positions on the Sudoku board. It finds the first empty position, tries numbers 1 through 9, and recursively explores valid possibilities. If a valid solution is found, it returns the solved board; otherwise, it backtracks and continues the search.

Returns:

- (vector of vectors or nil): The solved Sudoku board or nil if no solution is found.

## move-correct

(**move-correct** number row column sudoku)

Checks if the move is correct and updates the Sudoku board.

Args:

- number (integer): The number to be placed.

- row (integer): Row index.

- column (integer): Column index.

- sudoku (vector of vectors): The Sudoku grid.

Returns:

- (vector of vectors or nil): The updated Sudoku board if the move is correct; otherwise, nil.

# (ns sudoku.stopwatch)

## start-stopwatch

(**start-stopwatch**)

Function sets current time to the atom called time-counted using time function from java class Instant

## stop-stopwatch

(**stop-stopwatch**)

Function calculates time elapsed from the call of start-stopwatch till call of this function.

Returns:

-Integer: Time elapsed in milliseconds.

## format-duration

(**format-duration** duration-ms)

Function formats time in millisecond to a format HH:mm:ss

Args:

-duration-ms(milliseconds): time elapsed

Returns:

-String: Time elapsed in format HH:mm:ss

# (ns sudoku.api-generator)

## fetch-sudoku

(**fetch-sudoku** difficulty)

Function uses API call to get solved sudoku. Function gets different sudoku level based on the given parameter.

Args:

-difficulty (string): player entered

Returns:

-(vector of vectors or nil):Solved sudoku or nil if request failed.

## remove-positions

(**remove-positions** solved-sudoku difficulty-level)

Function removes certain amount of numbers from solved sudoku based on the given difficulty.

Args:

-difficulty(string): value defined by user

Returns:

-(vector of vectors): Initial sudoku that is ready for the game.

## get-new-generated-sudoku

(**get-new-generated-sudoku** difficulty-level)

This function calls two helper functions. One for fetching solved sudoku from API and the other one to remove certain amount of numbers from the grid based on the given difficulty.

Args:

- difficulty-level (string)

Returns:

-(vector of vectors): Initial sudoku that is ready for the game.

# (ns sudoku.database)

## format-date-to-string

(**format-date-to-string** date)

Function formats date gotten from the database and formats it to string in a format: HH:mm:ss

Args:

-date (java sql date)

Returns:

String: Time format HH:mm:ss

## execute-statements

(**execute-statements** statements)

This function is calling jdbc function – with-db-transaction for adding default rows inside H2 embedded database.

Args:

-statement: Vector of strings – SQL statements

Returns:

-Vector of integers: Containing zeroes and ones indicating execution of SQL statement.

## save-game

(**save-game** player-name difficulty time-elapsed])

Function saves an instance inside a database.

Args:

-player-name(String): Name that player entered.

-difficulty (String): Level of difficulty that player chosen at the beginning of the game.

-time-elapsed(String): Format HH:mm:ss.

Returns:

-(list of maps): Map contains key :id and it’s value that was auto generated from database.

## get-rankings-by-difficulty

(**get-rankings-by-difficulty** difficulty)

Function retrieves data (player name and time elapsed) by the given difficulty from the database and sorts data by time in ascending order.

Args:

-difficulty (string): difficulty level of the game

Returns:

-(list of maps): Every map contains next keywords :player\_name, :difficulty, :time\_elapsed. Every map represents a row from the database.

(**get-all-rankings**)

Function retrieves data (player name, difficulty and time elapsed) for all difficulties from the database and sorts data by time in ascending order.

Returns:

- (list of maps): Every map contains next keywords :player\_name, :difficulty, :time\_elapsed. Every map represents a row from the database

## print-rankings-by-difficulty

(**print-rankings-by-difficulty** difficulty)

Function formats and prints data from database that are retrieved by given difficulty.

Args:

- difficulty (string): difficulty level of the game

## custom-order

(**custom-order** data)

Function reorders given data by difficulty in next order: extreme easy, easy, medium, hard

Args:

-data: Map with keys that represent difficulty level.

Returns:

(map): Returns a map with reordered according to ordered-difficulties

## group-by-difficulty

(**group-by-difficulty** results)

This functions call helper function that groups ‘results’ by difficulty. Each element from ‘results’ is a map containing :difficulty key. Returns a map where kays are difficulty levels and values are collections of results corresponding to each difficulty level.

Args:

-results: list of maps – data from database

Returns:

-(map): Map where keys are values of difficulties . Value of each key is a vector that contains maps. Each mp is an instance from database.

## print-all-rankings

(print-all-rankings)

Function formats and prints data from database that are retrieved for all difficulties.

Args:

- difficulty (string): difficulty level of the game