

Posix-Nexus Javascript **JS**



Canine-Table

March 30, 2025

Contents

String	III
Integer	V
Object	VI
Types	VII
Class	IX
Event	X
Animation	XI
Array	XII



I String

I String

] The following functions enhance string processing capabilities, introducing features for sanitization, character extraction, and conditional formatting.

- ➔ **Str.join(val, str, sep = ")**: Concatenates the given string **str** to the value **val**, using the optional separator **sep**. Ensures **val** is defined before appending the separator and the string.
- ➔ **Str.merge(val1, val2, def1 = ' ', def2 = ' ', sep = ' ')**: Combines two values **val1** and **val2**, defaulting to **def1** and **def2** if undefined. Uses **sep** as the separator and relies on **Str.join** for the final concatenation.
- ➔ **Str.toLower(val)**: Converts the given value **val** to lowercase. Ensures **val** is defined before transformation. Returns the transformed string.
- ➔ **Str.toUpper(val)**: Converts the given value **val** to uppercase. Ensures **val** is defined before performing the transformation. Returns the transformed string.
- ➔ **Str.camelCase(val)**: Transforms the given string **val** into camel case format. Adjusts capitalization based on position and removes unnecessary spaces or special characters, ensuring the first letter is lowercase.
- ➔ **Str.titleCase(val)**: Converts the given string **val** into title case format. Capitalizes the first letter of each word and lowers the rest, separating words with spaces for consistent formatting.
- ➔ **Str.camelTitleCase(val)**: Converts a camel-case formatted string **val** into a readable title case format. Splits the string into words based on uppercase letters and ensures proper capitalization of the first word.
- ➔ **Str.joinCase(val, rpl = ", fnd = 's+')**: Joins segments of the given value **val** using a replacement string **rpl**. Splits the input based on the provided delimiter **fnd**, transforming each segment to lowercase before joining.
- ➔ **Str.kebabCase(val)**: Converts the given string **val** into kebab-case format. Uses **Str.joinCase** to replace spaces with hyphens and ensures proper formatting.
- ➔ **Str.snakeCase(val)**: Converts the given string **val** into snake-case format by replacing spaces or delimiters with underscores (**_**). Uses **Str.joinCase** for consistent processing.
- ➔ **Str.remove(val, mth = 'null|undefined')**: Cleans the given string **val** by removing matches for the pattern **mth**, replacing them with spaces. Normalizes and trims the result to ensure no excess whitespace remains.





^ I String

- ➔ **Str.upper()**: Returns a string containing all uppercase English alphabet letters (A–Z). Does not require any input and ensures consistent output.
- ➔ **Str.lower()**: Returns a string containing all lowercase English alphabet letters (a–z). Does not require any input and ensures consistent output.
- ➔ **Str.digit()**: Returns a string containing all numeric digits (0–9). Designed for easy inclusion in character sets.
- ➔ **Str.xdigit(val)**: Generates a string of hexadecimal characters based on the given value **val**. Includes lowercase (a–f), uppercase (A–F), and digits (0–9). Adjusts the inclusion of uppercase or lowercase letters based on truthiness or falsiness of **val**.
- ➔ **Str.punct()**: Returns a string containing common punctuation characters, including symbols such as !, #, @, and more.
- ➔ **Str.alpha()**: Generates a string containing all alphabetic characters, combining uppercase (A–Z) and lowercase (a–z) alphabets.
- ➔ **Str.alnum()**: Combines alphabetic (**Str.alpha()**) and numeric (**Str.digit()**) characters into a single string.
- ➔ **Str.graph()**: Produces a string containing all printable graphical characters, combining alphanumeric (**Str.alnum()**) and punctuation (**Str.punct()**) characters.
- ➔ **Str.random(val, chars = 'alnum', sep = ',')**: Generates a random string of length **val** using the specified character sets **chars**. Supports various character options such as `alnum`, `digit`, `graph`, `lower`, and more. Allows customization by splitting character definitions with the provided separator **sep**.
- ➔ **Str.implode(val, sep = ',')**: Joins the elements of the array **val** into a single string, using the optional separator **sep**. Returns an empty string if **val** is not an array.
- ➔ **Str.sanitize(val)**: Cleans the given string **val** by removing all non-alphanumeric characters (a–z, A–Z, 0–9). Returns the sanitized string.
- ➔ **Str.lastChar(str, val)**: Extracts the portion of the string **str** following the last occurrence of the specified value **val**. Ensures both inputs are defined before performing the operation. Returns the resulting substring or `undefined` if **val** is not found.
- ➔ **Str.conditional(val, pre = ", post = ")**: Constructs a new string by appending the prefix **pre** and suffix **post** around the value **val**. Returns an empty string if **val** is not defined.



II Integer

II Integer

] The following functions provide advanced numerical utilities for iterative looping, directional number generation, and proportional value distribution.

- ➔ **Int.loop(num, start = 0, stop):** Adjusts the numerical value **num** to fit within the inclusive range defined by **start** and **stop**. Handles cyclic behavior, ensuring values outside the range wrap back inwards. Returns 0 if **num** is not a valid float.
- ➔ **Int.direction(start = 0, stop, skip = 1):** Generates a sequence of numbers between **start** and **stop**, incrementing or decrementing by **skip**. Supports infinite iteration when **stop** is not defined. Returns a generator object.
- ➔ **Int.distribute(low, up, range):** Calculates the proportional distribution of a value within the range defined by **low** and **up**, adjusted by the given **range**. Returns the ceiling of the resulting value for accurate interval placement.
- ➔ **Int.randomRange(stop, start = 0):** Generates a random integer within the inclusive range specified by **start** and **stop**. Ensures both bounds are integral values, adjusting behavior with **Int.loop** for proper range wrapping.
- ➔ **Int.wholeRandom():** Produces a whole random number derived from the decimal portion of a `Math.random()` call. Uses **Str.lastChar()** to extract the desired numeric portion.
- ➔ **Int.odd(num, val):** Adjusts the given number **num** to the nearest odd integer. If **val** is truthy, increases **num** to the next odd value if necessary. If falsy, decreases **num** to the previous odd value if required.
- ➔ **Int.isPos(val):** Determines if the given value **val** is positive or zero. Uses `Math.sign()` for precise validation. Returns `true` for positive values or zero.
- ➔ **Int.isNeg(val):** Determines if the given value **val** is negative. Uses `Math.sign()` for accurate validation. Returns `true` for negative values, including `-0`.



III Object

III Object

] The following functions provide powerful object manipulation utilities, enabling reflection, property checking, and dynamic assignment operations for JavaScript objects.

- ➔ **Obj.reflect(val)**: Inspects the given value **val** and retrieves the keys of its prototype if it is an object. If the prototype contains only one property, retrieves the object's own keys instead.
- ➔ **Obj.methods(val)**: Logs all methods of the given object **val**, excluding common non-functional properties such as `length`, `name`, `prototype`, and others.
- ➔ **Obj.isProp(val, prop)**: Checks if the given property **prop** exists within the reflected properties of the value **val**. Returns `true` if found, otherwise `false`.
- ➔ **Obj.assign(val, prop, obj)**: Dynamically assigns properties from **obj** to **val**. If **prop** is defined, assigns properties to the specific sub-object at **val[prop]**; otherwise, assigns directly to **val**.



IV Types

IV Types

] The following functions extend type-checking and utility capabilities, enabling validation for defined values and array-like structures, including dynamic handling of default values.

- ➔ **Type.isObject(val)**: Determines whether the given value **val** is a non-null object and not an array. Returns `true` if the value meets these criteria, otherwise `false`.
- ➔ **Type.isFunction(val)**: Verifies whether the given value **val** is a function. Returns `true` if the value is callable, otherwise `false`.
- ➔ **Type.isClass(val)**: Checks if the provided value **val** is a class definition. Returns `true` if the value is a class, otherwise `false`. Utilizes a regex check to distinguish class structures.
- ➔ **Type.isFloat(val)**: Determines whether the given value **val** is a valid floating-point number. Supports scientific notation (e.g., `1.23e4`) and returns `true` for valid inputs, otherwise `false`.
- ➔ **Type.isIntegral(val)**: Verifies whether the given value **val** is a valid integer (including negatives and positives). Returns `true` if the value meets this criteria, otherwise `false`.
- ➔ **Type.isString(val)**: Checks if the given value **val** is a string. Returns `true` if **val** is of type `string` or an instance of `String`, otherwise `false`.
- ➔ **Type.isBoolean(val)**: Determines if the given value **val** is a boolean. Returns `true` if **val** is of type `boolean` or an instance of `Boolean`, otherwise `false`.
- ➔ **Type.isEmpty(val)**: Checks whether the given value **val** is an empty string (`""`). Returns `true` if **val** is empty, otherwise `false`.
- ➔ **Type.isTrue(val)**: Validates whether the given value **val** loosely equals `true`. Returns `true` for truthy values, otherwise `false`.
- ➔ **Type.isFalse(val)**: Validates whether the given value **val** loosely equals `false`. Returns `true` for falsy values, otherwise `false`.
- ➔ **Type.isUndefined(val)**: Checks if the given value **val** is explicitly undefined. Returns `true` if **val** equals `undefined`, otherwise `false`.
- ➔ **Type.isNull(val)**: Determines if the given value **val** is explicitly `null`. Returns `true` if **val** equals `null`, otherwise `false`.



^ IV Types

- ➔ **Type.isAbsolute(val)**: Validates whether the given value **val** is an absolute number (equal to its absolute value). Returns `true` for absolute numbers, otherwise `false`.
- ➔ **Type.isJson(val)**: Checks if the given value **val** is a valid JSON string. Returns `true` if **val** can be successfully parsed by `JSON.parse`, otherwise `false`.
- ➔ **Type.isDefined(val, def)**: Checks if the given value **val** is defined (not `undefined` or an empty string). Returns the default value **def** if **val** is not defined, otherwise returns `true` or the value of **val** if **def** is specified.
- ➔ **Type.isArray(val, def)**: Determines whether the given value **val** is an array or a `Set`. If true, returns **val** or `true` depending on **def**. If **val** is a defined string (but not an object), splits it into an array using **def** as the delimiter. Returns an empty array or `false` for other cases, depending on **def**.



V Class

V Class

] The following functions offer versatile utilities for managing and manipulating object classes, enabling default property handling, aliasing, and hierarchical navigation.

- ➔ **nexClass.defaults(def, obj, mth)**: Assigns default properties from **def** to the given object **obj**. Utilizes **mth** for method-based value matching. Removes properties with undefined or `false` values while populating the `obj.prop` structure.
- ➔ **nexClass.prop(ref, obj, cls)**: Maps properties from the reference array **ref** to the given object **obj**. Supports nested property handling and invokes class-specific functions on matching elements when the class **cls** is provided.
- ➔ **nexClass.expand(val, obj)**: Expands the object **val** into **obj** by converting array values into key-value mappings, preserving object relationships.
- ➔ **nexClass.cherryPick(obj)**: Randomly selects and returns an element from the array **obj**. Uses `Int.loop()` for cyclic wrapping of the index.
- ➔ **nexClass.alias(als, opt, obj)**: Applies alias mappings defined in **als** to the object **obj**, replacing keys with their corresponding aliases. Supports optional value matching using **opt**. Updates the object structure dynamically.
- ➔ **nexClass.back(obj, num = 1)**: Recursively traverses the parent hierarchy of the object **obj**, moving up by **num** levels. Returns the resulting ancestor object or the original **obj** if no further traversal is possible.



VI Event

VI Event

] The following functions provide advanced event handling utilities, enabling robust management of DOM events, mutations, overflow detection, and controlled execution timing.

- ➔ **nexEvent.add(obj)**: Adds event listeners to a specified DOM node. Maps shorthand event identifiers to their full names (e.g., 'c' to 'click') and applies the associated action for each event. Validates event and node details before execution.
- ➔ **nexEvent.mutation(ml, obs)**: Handles mutation events based on a list **ml**. Logs changes to child nodes or attributes, such as additions, removals, or modifications, with detailed outputs.
- ➔ **nexEvent.overflow(elm)**: Determines whether the content of the given element **elm** exceeds its visible dimensions, checking both height and width overflow. Returns `true` if overflow exists, otherwise `false`.
- ➔ **nexEvent.throttle(func, limit)**: Restricts the execution frequency of the given function **func** to once every **limit** milliseconds. Ensures proper spacing of executions, even during rapid triggering.
- ➔ **nexEvent.debounce(func, delay)**: Delays the execution of the given function **func** until **delay** milliseconds have passed since the last invocation. Helps reduce redundant calls during high-frequency triggers.
- ➔ **nexEvent.delay(ms)**: Creates a delay for the specified number of milliseconds **ms**. Returns a promise that resolves after the delay, allowing asynchronous control.



VII Animation

VII Animation

] The following functions enhance user interface customization by dynamically managing themes and integrating responsive design preferences.

- ➔ **nexAnime.setTheme(val)**: Adjusts the current theme based on the specified value **val**. If no value is provided, determines the theme by checking session storage, user preference, or system settings. Ensures the theme is stored and updated by applying the relevant attribute to the root element using **nexNode.SetAttr**. Returns the selected theme.





VIII Array

VIII Array

] The following functions provide advanced utilities for array manipulation, enabling operations like uniqueness setting, querying, range generation, and structure flattening.

- ➔ **Arr.add(val, arr, sep = ',')**: Combines the given value **val** and array **arr** into a single array. Converts both inputs into array structures using the specified separator **sep** before concatenation.
- ➔ **Arr.search(val, arr, act, sep = ',')**: Searches the array **arr** for elements matching the given value **val**, based on the specified action **act**. Supports methods like `includes`, `endsWith`, and `startsWith`, with an optional separator **sep** for input transformation.
- ➔ **Arr.start(val, arr, sep)**: Filters the array **arr** for elements that start with the given value **val**. Acts as a shorthand for **Arr.search()** with a default action of `startsWith`.
- ➔ **Arr.end(val, arr, sep)**: Filters the array **arr** to find elements ending with the given value **val**. Acts as a shorthand for **Arr.search()** with a default action of `endsWith`.
- ➔ **Arr.within(val, arr, sep)**: Filters the array **arr** to find elements that include the given value **val**. Serves as a shorthand for **Arr.search()** with a default action of `includes`.
- ➔ **Arr.in(val, arr, sep = ',')**: Checks whether the given value **val** exists within the array **arr**. Returns `true` if found, otherwise `false`.
- ➔ **Arr.lengths(val, sep = ',', act)**: Determines the length of the longest or shortest element in the array **val**, based on the **act** parameter. Returns 0 if **val** is not defined.
- ➔ **Arr.lengthMatch(val, sep, len)**: Filters the array **val** to find elements with lengths matching the specified **len**. Returns an array of matched elements.
- ➔ **Arr.extremeMatch(val, sep = ',', ext)**: Finds elements in the array **val** with the shortest or longest length, based on the **ext** parameter. If multiple elements match, returns them all; otherwise, returns the singular match.
- ➔ **Arr.shortest(val, sep = ',')**: Identifies the shortest element in the array **val**. Uses **Arr.extremeMatch()** for streamlined processing. Returns the shortest value or an array of matches.
- ➔ **Arr.longest(val, sep = ',')**: Identifies the longest element in the given array **val**. Uses **Arr.extremeMatch()** to determine the result, returning either a singular match or multiple matching elements.



^ VIII Array

- ➔ **Arr.explode(val)**: Splits the given string **val** into an array of individual characters. Returns an empty array if **val** is not defined.
- ➔ **Arr.difference(arrA, arrB, side, sep = ',')**: Computes the difference between two arrays, **arrA** and **arrB**. Returns elements unique to either array. The **side** parameter controls whether differences are calculated unidirectionally or bidirectionally.
- ➔ **Arr.left(arrA, arrB, sep = ',')**: Computes elements in **arrA** that are not present in **arrB**. Acts as a shorthand for **Arr.difference()** with a unidirectional configuration.
- ➔ **Arr.right(arrA, arrB, sep = ',')**: Computes elements in **arrB** that are not present in **arrA**. Acts as a shorthand for **Arr.difference()** with reversed inputs.
- ➔ **Arr.shortStart(val, opt, sep = ',', def)**: Searches for the shortest matching element within **opt** starting with **val**. Returns a default value **def** if no match or multiple matches are found.
- ➔ **Arr.rotate(arr, steps, sep = ',')**: Rotates the elements of the array **arr** by the specified number of **steps**. Wraps elements cyclically to maintain array integrity.
- ➔ **Arr.set(obj, sep = ',')**: Removes duplicate entries from the given input **obj** and returns an array of unique values. Processes the input as an array using the separator **sep**.
- ➔ **Arr.query(val, obj, sep = ',')**: Generates values from **val** based on the given **obj**. Supports both array and object formats for **obj**, providing flexible start, stop, and skip options. Returns a generator object.
- ➔ **Arr.range(val, bound, skip)**: Creates a range of numbers starting from **bound** to **val**, with optional skipping intervals. Adjusts the input format for boundary conditions using special characters like '<' and '>'.
- ➔ **Arr.expandBoolean(obj)**: Expands a boolean configuration object, converting **true** and **false** arrays into corresponding key-value pairs. Merges the expanded entries back into the original object for consistent usage.
- ➔ **Arr.flatten(obj, ref, sep = '-', osep)**: Flattens nested structures, generating a single-level array. Uses the reference **ref** for key transformation and applies the separator **sep** to construct hierarchical paths. Optionally joins the result using **osep**.
- ➔ **Arr.types(arr, type, not = false)**: Filters the array **arr** based on the specified data types in **type**. Supports exclusion when **not** is set to **true**.
- ➔ **Arr.prePostAppend(arr, pre = "", post = "", sep = ',')**: Appends the specified prefix **pre** and suffix **post** to each element of the array **arr**. Joins the results using the separator **sep**.