In pure JS	
Pure JS	Lodash

function assign(target, ...sources) {
  if (target == null) {
    throw new TypeError('Cannot convert u
  }
 	
  let to = Object(target);
 	
  for (let source of sources) {
    if (source != null) {
      for (let key in source) {
        if (Object.prototype.hasOwnProperty
          to[key] = source[key];
        }
      }
    }
  }
 	
  return to;
}
	assign

Object.assign	assignIn

function bind(func, thisArg, ...partials) {
  return function(...args) {
    return func.call(thisArg, ...partials, ...arg
  }
}
	bind

function camelCase(string) {
  const words = string.match(/[A-Za-z0-9]
  const camelCaseWords = words.map((w
    if (index === 0) {
      return word.toLowerCase();
    }
    return word.charAt(0).toUpperCase() +
  });
  return camelCaseWords.join('');
}
	camelCase

const capitalize = str => `${str.charAt(0).t	capitalize

In pure JS	

function chain(value) {
  const result = {value};
 	
  const lodashMethods = {
    // Define each Lodash method as a fun
    map: (callback) => {
      result.value = result.value.map(callba
      return lodashMethods;
    },
    filter: (callback) => {
      result.value = result.value.filter(callba
      return lodashMethods;
    },
    reduce: (callback, initialValue) => {
      result.value = result.value.reduce(call
      return lodashMethods;
    },
    // Add more Lodash methods as neede
  };
 	
  // Return the Lodash method object, whi
  return lodashMethods;
}
	chain

function clone(value) {
  // Check if the value is an array
  if (Array.isArray(value)) {
    // Use the spread operator to create a s
    return [...value];
  }
 	
  // Check if the value is an object
  if (typeof value === "object" && value !=
    // Use Object.assign to create a shallov
    return Object.assign({}, value);
  }
 	
  // For all other types of values, return the
  return value;
}
	clone

In pure JS	
function cloneDeep(value) {	
// Check if the value is an array	
if (Array.isArray(value)) {	
// Use the map method to create a new	
return value.map(cloneDeep);	
}	
// Check if the value is an object	
if (typeof value === "object" && value !=	
// Create a new object to hold the clone	
const clonedObject = {};	
// Recursively clone each property of th	
for (const key in value) {	
clonedObject[key] = cloneDeep(value	
}	
return clonedObject;	
}	
// For all other types of values, return the	
return value;	
}	cloneDeep
function cloneDeepWith(value, customize	
// Check if the customizer is a function	
if (typeof customizer !== "function") {	
customizer = undefined;	
}	
// Check if the value is an array	
if (Array.isArray(value)) {	
// Use the map method to create a new	
return value.map(element => {	
return cloneDeepWith(element, custo	
});	
}	
// Check if the value is an object	
if (typeof value === "object" && value !=	
// Create a new object to hold the clone	
const clonedObject = {};	
// Recursively clone each property of th	
for (const key in value) {	
const clonedValue = cloneDeepWith(	
clonedObject[key] = customizer ? cus	
}	
return clonedObject;	
}	
// For all other types of values, return the	
return customizer ? customizer(value) :	
}	cloneDeepWith
Array.prototype.filter(Boolean)	compact
const constant = x => () => x;	constant

In pure JS	

function countBy(collection, iteratee) {
  // Create an empty object to hold the res
  const result = {};
 	
  // If the iteratee is a function, use it to ma
  if (typeof iteratee === "function") {
    collection = collection.map(iteratee);
  } else if (typeof iteratee === "string") {
    // If the iteratee is a string, use it as a p
    collection = collection.map(element =>
  }
 	
  // Loop through the mapped collection ar
  for (const element of collection) {
    result[element] = (result[element] || 0) +
  }
 	
  return result;
}
	countBy

function debounce(func, wait, options) {
  let timeoutId;
 	
  return function debounced(...args) {
    const context = this;
 	
    clearTimeout(timeoutId);
 	
    timeoutId = setTimeout(() => {
      func.apply(context, args);
    }, wait);
  };
}
	debounce

const defaults = (...args) => args.reverse	defaults

function defaultsDeep(target, ...sources) {
  for (const source of sources) {
    if (typeof source === "object" && sourc
      for (const key in source) {
        if (Object.prototype.hasOwnProperty
          if (typeof target[key] === "object" &
            defaultsDeep(target[key], source[k
          } else if (!(key in target)) {
            target[key] = source[key];
          }
        }
      }
    }
  }
 	
  return target;
}
	defaultsDeep

function delay(func, wait, ...args) {
  setTimeout(() => {
    func(...args);
  }, wait);
}
	delay

Use Array.prototype.filter and Array.protot	difference

In pure JS	
function drop(array, n = 1) { if (!Array.isArray(array) || array.length == return []; } return array.slice(n); }	drop
function dropRight(array, n = 1) { if (!Array.isArray(array) || array.length == return []; } return array.slice(0, -n); }	dropRight
Array.prototype.forEach	each
function escape(string) { const regex = /[&<>"'\/]/g; const escapeChars = { '&': '&amp;', '<': '&lt;', '>': '&gt;', '"': '&quot;', "'": '&#x27;', '/': '&#x2F;' }; return string.replace(regex, match => es }	escape
function escapeRegExp(string) { const regex = /[\\^$.*+?()[\]{}\|]/g; return string.replace(regex, '\\$&'); }	escapeRegExp
Array.prototype.every	every
function extend(target, ...sources) { for (const source of sources) { for (const key in source) { if (Object.prototype.hasOwnProperty.c target[key] = source[key]; } } } return target; }	extend
function fill(array, value, start = 0, end = a for (let i = start; i < end; i++) { array[i] = value; } return array; }	fill
Array.prototype.filter	filter
Array.prototype.find	find

In pure JS	

function findIndex(array, predicate, fromIn
  for (let i = fromIndex; i < array.length; i+
    if (predicate(array[i], i, array)) {
      return i;
    }
  }
  return -1;
}
	findIndex

function findKey(object, predicate) {
  for (let key in object) {
    if (object.hasOwnProperty(key) && pre
      return key;
    }
  }
  return undefined;
}
	findKey

function findLast(collection, predicate, fro
  for (let i = fromIndex; i >= 0; i--) {
    if (predicate(collection[i], i, collection)) {
      return collection[i];
    }
  }
  return undefined;
}
	findLast

function findLastIndex(array, predicate, fro
  for (let i = fromIndex; i >= 0; i--) {
    if (predicate(array[i], i, array)) {
      return i;
    }
  }
  return -1;
}
	findLastIndex

array[0]	first
Array.prototype.flat	flatten

function flattenDeep(array) {
  return array.reduce((acc, val) => Array.is
}
	flattenDeep

function flow(...funcs) {
  return function(...args) {
    let result = funcs[0](...args);
    for (let i = 1; i < funcs.length; i++) {
      result = funcs[i](result);
    }
    return result;
  }
}
	flow

In pure JS	

function forEach(collection, iteratee) {
  if (Array.isArray(collection)) {
    for (let i = 0; i < collection.length; i++) {
      iteratee(collection[i], i, collection);
    }
  } else {
    for (let key in collection) {
      if (Object.prototype.hasOwnProperty.c
        iteratee(collection[key], key, collectio
      }
    }
  }
  return collection;
}
	forEach

function forEachRight(collection, iteratee)
  if (Array.isArray(collection)) {
    for (let i = collection.length - 1; i >= 0; i-
      iteratee(collection[i], i, collection);
    }
  } else {
    const keys = Object.keys(collection);
    for (let i = keys.length - 1; i >= 0; i--) {
      const key = keys[i];
      if (Object.prototype.hasOwnProperty.c
        iteratee(collection[key], key, collectio
      }
    }
  }
  return collection;
}
	forEachRight

function forOwn(object, iteratee) {
  for (const key in object) {
    if (object.hasOwnProperty(key)) {
      iteratee(object[key], key, object);
    }
  }
  return object;
}
	forOwn

function fromPairs(pairs) {
  const result = {};
  for (let i = 0; i < pairs.length; i++) {
    result[pairs[i][0]] = pairs[i][1];
  }
  return result;
}
	fromPairs

function get(object, path, defaultValue) {
  const keys = Array.isArray(path) ? path :
  let result = object;
  for (let i = 0; i < keys.length; i++) {
    const key = keys[i];
    result = result[key];
    if (result === undefined) {
      return defaultValue;
    }
  }
  return result;
}
	get

In pure JS	
same as get	get as getField
function groupBy(collection, iteratee) { const groups = {}; for (let i = 0; i < collection.length; i++) { const key = iteratee(collection[i]); if (groups[key] === undefined) { groups[key] = []; } groups[key].push(collection[i]); } return groups; }	groupBy
function gt(value, other) { return value > other; }	gt
function gte(value, other) { return value >= other; }	gte
function has(object, path) { return object != null && Object.prototype }	has
function hasIn(object, path) { let currentObj = object; for (const key of path.split('.')) { if (currentObj != null && key in currentC currentObj = currentObj[key]; } else { return false; } } return true; }	hasIn
function head(array) { return (array != null && array.length) ? array[0] : undefined; }	head
const identity = x => x;	identity
Array.prototype.includes or String.prototyp	includes
function indexOf(array, value) { if (array == null) { return -1; } const length = array.length; for (let i = 0; i < length; i++) { if (array[i] === value) { return i; } } return -1; }	indexOf

In pure JS	
function intersection(...arrays) { if (!arrays || !arrays.length) { return []; } const result = []; const firstArray = arrays[0]; const length = firstArray.length; for (let i = 0; i < length; i++) { const value = firstArray[i]; if (result.includes(value)) { continue; } if (arrays.every(array => array.includes(result.push(value); } } return result; }	intersection
Array.isArray	isArray
function isBoolean(value) { return typeof value === 'boolean'; }	isBoolean
function isDate(value) { return value instanceof Date && !isNaN(}	isDate
function isEmpty(value) { if (value == null) { return true; } if (typeof value === 'string' || Array.isArra return !value.length; } if (typeof value === 'object') { return !Object.keys(value).length; } return false; }	isEmpty

In pure JS	

```
function isEqual(value, other) {
  // Get the value type
  const type = Object.prototype.toString.ca

  // If the two values are not of the same ty
  if (type !== Object.prototype.toString.call
    return false;
  }

  // If the value is a primitive type, do a sim
  if (['[object Number]', '[object String]', '[ob
    return value === other;
  }

  // If the value is a function, check that the
  if (type === '[object Function]') {
    return value.toString() === other.toStrin
  }

  // If the value is an object, perform a dee
  if (type === '[object Object]') {
    const keys = Object.keys(value);

    if (keys.length !== Object.keys(other).le
      return false;
    }

    for (let i = 0; i < keys.length; i++) {
      const key = keys[i];

      if (!other.hasOwnProperty(key) || !isEq
        return false;
      }
    }

    return true;
  }

  // If the value is an array, perform a deep
  if (type === '[object Array]') {
    if (value.length !== other.length) {
      return false;
    }

    for (let i = 0; i < value.length; i++) {
      if (!isEqual(value[i], other[i])) {
        return false;
      }
    }

    return true;
  }

  // If the value is a Date object, perform a
  if (type === '[object Date]') {
    return value.getTime() === other.getTim
  }

  return false;
}
```

isEqual

In pure JS	

function isFunction(value) {
  return typeof value === 'function';
}
	isFunction

function isNaN(value) {
  return Number.isNaN(value);
}
	isNaN

function isNull(value) {
  return value === null;
}
	isNull

function isNumber(value) {
  return typeof value === 'number' && isFi
}
	isNumber

function isObject(value) {
  const type = typeof value;
  return value !== null && (type === 'objec
}
	isObject

same as isObject	isObject as isObjectLodash

function isPlainObject(value) {
  if (typeof value !== 'object' || value === n
    return false;
  }
 	
  const proto = Object.getPrototypeOf(valu
  if (proto === null) {
    return true;
  }
 	
  let baseProto = proto;
  while (Object.getPrototypeOf(baseProto)
    baseProto = Object.getPrototypeOf(bas
  }
 	
  return proto === baseProto;
}
	isPlainObject

function isString(value) {
  return typeof value === 'string' || value ir
}
	isString

function isUndefined(value) {
  return typeof value === 'undefined';
}
	isUndefined

In pure JS	

function iteratee(value) {
  if (typeof value == 'function') {
    return value;
  }
  if (Array.isArray(value)) {
    return function (obj) {
      return obj[value[0]] === value[1];
    };
  }
  if (typeof value == 'object') {
    return function (obj) {
      for (var key in value) {
        if (obj[key] !== value[key]) {
          return false;
        }
      }
      return true;
    };
  }
  return function (obj) {
    return obj[value];
  };
}
	iteratee

function keyBy(array, keyFunction) {
  return array.reduce((result, element) =>
    const key = keyFunction(element);
    result[key] = element;
    return result;
  }, {});
}
	keyBy

Object.keys	keys
arr[-1]	last

function lt(value, other) {
  return value < other;
}
	lt

function lte(value, other) {
  return value <= other;
}
	lte

Array.prototype.map	map

function mapKeys(obj, fn) {
  return Object.fromEntries(
    Object.entries(obj).map(([key, val]) => [
  );
}
	mapKeys

function mapValues(obj, iteratee) {
  const result = {};
  for (const [key, value] of Object.entries(
    result[key] = iteratee(value, key, obj);
  }
  return result;
}
	mapValues

In pure JS	

function max(array) {
  if (!Array.isArray(array) || array.length ==
    return undefined;
  }
 	
  let maxValue = array[0];
 	
  for (let i = 1; i < array.length; i++) {
    if (array[i] > maxValue) {
      maxValue = array[i];
    }
  }
 	
  return maxValue;
}
	max

function memoize(fn) {
  const cache = {};
  return function(...args) {
    const key = JSON.stringify(args);
    if (cache.hasOwnProperty(key)) {
      return cache[key];
    } else {
      const result = fn.apply(this, args);
      cache[key] = result;
      return result;
    }
  };
}
	memoize

function merge() {
  const result = {};
  for (let i = 0; i < arguments.length; i++) {
    const obj = arguments[i];
    if (!obj) continue;
    for (let key in obj) {
      if (obj.hasOwnProperty(key)) {
        if (Object.prototype.toString.call(obj[
          result[key] = merge(result[key], obj[
        } else {
          result[key] = obj[key];
        }
      }
    }
  }
  return result;
}
	merge

In pure JS	

function mergeWith(...objects) {
  // Define the customizer function
  function customizer(objValue, srcValue)
    // If the value is an array, concatenate t
    if (Array.isArray(objValue) && Array.isA
      return objValue.concat(srcValue);
    }
    // If the value is an object, call the merg
    if (typeof objValue === "object" && type
      return mergeWith(objValue, srcValue)
    }
    // If none of the above, use the default l
    return undefined;
  }
  // Use the spread operator and Object.as
  return Object.assign({}, ...objects, custor
}
	mergeWith

function min(collection) {
  return Math.min(...collection);
}
	min

const noop = () => undefined;	noop

function omit(obj, props) {
  return Object.keys(obj)
    .filter((key) => !props.includes(key))
    .reduce((acc, key) => {
      acc[key] = obj[key];
      return acc;
    }, {});
}
	omit

function omitBy(obj, predicate) {
  return Object.entries(obj).reduce((acc, [k
    if (!predicate(value)) {
      acc[key] = value;
    }
    return acc;
  }, {});
}
	omitBy

function once(func) {
  let result;
  let hasBeenCalled = false;
 	
  return function() {
    if (!hasBeenCalled) {
      hasBeenCalled = true;
      result = func.apply(this, arguments);
    }
    return result;
  }
}
	once

In pure JS	

function orderBy(arr, props, orders) {
  // If the properties or orders array is not p
  props = props || [];
  orders = orders || [];
 	
  // If props is not an array, convert it into a
  if (!Array.isArray(props)) {
    props = [props];
  }
 	
  // If orders is not an array, convert it into
  if (!Array.isArray(orders)) {
    orders = [orders];
  }
 	
  // Return the sorted array
  return arr.sort(function (a, b) {
    for (var i = 0; i < props.length; i++) {
      var prop = props[i];
      var order = orders[i] || 'asc';
      var aValue = a[prop];
      var bValue = b[prop];
 	
      if (aValue < bValue) {
        return order === 'asc' ? -1 : 1;
      } else if (aValue > bValue) {
        return order === 'asc' ? 1 : -1;
      }
    }
 	
    return 0;
  });
}
	orderBy

const padEnd = (str, len, pad = ' ') => str + pad.repeat(len - str.length).slice(0,	padEnd
const padStart = (str, len, pad = ' ') => pad.repeat(len - str.length).slice(0, Math	padStart
parseInt	parseInt

function partial(fn, ...args) {
  return function(...remainingArgs) {
    return fn(...args, ...remainingArgs);
  };
}
	partial

function partition(arr, predicate) {
  return arr.reduce(
    (result, current) => {
      result[predicate(current) ? 0 : 1].push(
      return result;
    },
    [[], []]
  );
}
	partition

In pure JS	

function pick(obj, keys) {
  return keys.reduce((acc, key) => {
    if (obj[key] !== undefined) {
      acc[key] = obj[key];
    }
    return acc;
  }, {});
}
	pick

function pickBy(obj, predicate) {
  const newObj = {};
  for (const key in obj) {
    if (Object.prototype.hasOwnProperty.ca
      const value = obj[key];
      if (predicate(value, key)) {
        newObj[key] = value;
      }
    }
  }
  return newObj;
}
	pickBy

function pluck(collection, property) {
  return collection.map(obj => obj[property
}
	pluck

_.prototype is a special property of the loc	prototype

function pull(array, ...values) {
  const length = array == null ? 0 : array.le
  const result = [];
 	
  for (let i = 0; i < length; i++) {
    const value = array[i];
    if (!values.includes(value)) {
      result.push(value);
    }
  }
 	
  array.length = 0;
  for (let i = 0, resultLength = result.length
    array[i] = result[i];
  }
 	
  return array;
}
	pull

function random(min = 0, max = 1, floatin
  if (floating) {
    return Math.random() * (max - min) + m
  }
  return Math.floor(Math.random() * (max -
}
	random

In pure JS	

function range(start, end, step = 1) {
  if (typeof end === 'undefined') {
    end = start;
    start = 0;
  }
  const result = [];
  let i = start;
  while ((step > 0 && i < end) || (step < 0 &
    result.push(i);
    i += step;
  }
  return result;
}
	range

Array.prototype.reduce	reduce

function reduceRight(array, iteratee, accu
  let i = array.length - 1;
  if (accumulator === undefined) {
    accumulator = array[i];
    i--;
  }
  for (; i >= 0; i--) {
    accumulator = iteratee(accumulator, ar
  }
  return accumulator;
}
	reduceRight

function reject(array, predicate) {
  const result = [];
  for (let i = 0; i < array.length; i++) {
    if (!predicate(array[i])) {
      result.push(array[i]);
    }
  }
  return result;
}
	reject

function remove(array, predicate) {
  let index = -1;
  const length = array == null ? 0 : array.le
  let resIndex = 0;
  const result = [];
 	
  while (++index < length) {
    const value = array[index];
    if (predicate(value, index, array)) {
      result[resIndex++] = value;
    }
  }
  index = -1;
  while (++index < length) {
    const value = array[index];
    if (!predicate(value, index, array)) {
      array[resIndex++] = value;
    }
  }
  array.length = resIndex;
  return result;
}
	remove

In pure JS	
function repeat(str, n) { return Array(n+1).join(str); }	repeat
function rest(array, n = 1) { if (!Array.isArray(array)) { throw new TypeError('Expected an arra } if (typeof n !== 'number') { throw new TypeError('Expected a numb } return array.slice(n); }	rest
const round = (num, precision) => { cons	round
function sample(array) { const length = array == null ? 0 : array.le return length ? array[Math.floor(Math.rar }	sample
function sampleSize(array, n) { const result = []; const length = array == null ? 0 : array.le if (!length || n < 1) { return result; } let index = -1; let lastIndex = length - 1; while (++index < n) { const rand = index + Math.floor(Math.ra result[index] = array[rand]; array[rand] = array[index]; } return result.slice(0, n); }	sampleSize

In pure JS	

function set(object, path, value) {
  // convert path string to an array of path keys
  const keys = path.split('.');
  // get the last key of the path
  const lastKey = keys.pop();
  // iterate over keys to get the nested object
  let nestedObj = object;
  for (const key of keys) {
    if (!nestedObj[key]) {
      nestedObj[key] = {};
    }
    nestedObj = nestedObj[key];
  }
  // set the value at the last key of the path
  nestedObj[lastKey] = value;
  return object;
}
 	
// example usage
const obj = { a: { b: { c: 1 } } };
set(obj, 'a.b.c', 2); // { a: { b: { c: 2 } } }	

function shuffle(array) {
  let currentIndex = array.length;
  let temporaryValue, randomIndex;
 	
  while (0 !== currentIndex) {
    randomIndex = Math.floor(Math.randon
    currentIndex -= 1;
 	
    temporaryValue = array[currentIndex];
    array[currentIndex] = array[randomInde
    array[randomIndex] = temporaryValue;
  }
 	
  return array;
}
	shuffle

function size(obj) {
  return Object.keys(obj).length;
}
	size

function snakeCase(str) {
  return str
    .replace(/[A-Z]/g, (match, offset) => (off
    .replace(/[\s\-]+/g, '_')
    .replace(/[^a-z0-9_]+/gi, '');
}
	snakeCase

Array.prototype.some	some

In pure JS	

function sortBy(array, callback) {
  return array.map((item, index) => ({
    value: item,
    index: index,
    criteria: callback(item, index),
  }))
    .sort((a, b) => {
      let criteriaA = a.criteria
      let criteriaB = b.criteria
 	
      if (criteriaA !== criteriaB) {
        if (criteriaA > criteriaB || criteriaA ===
          return 1
        } else if (criteriaA < criteriaB || criteria
          return -1
        }
      }
      return a.index - b.index
    })
    .map((item) => item.value)
}
	sortBy

function startCase(str) {
  return str
    .replace(/[^a-z0-9]+/gi, " ")
    .trim()
    .split(" ")
    .map(word => word[0].toUpperCase() +
    .join(" ");
}
	startCase

function startsWith(str, substr) {
  return str.slice(0, substr.length) === subs
}
	startsWith

Array.prototype.reduce((acc, num) => { a	sum

function take(array, n=1) {
  if (!Array.isArray(array)) {
    return [];
  }
  return array.slice(0, n);
}
	take

function template(str) {
  return function(data) {
    let result = str;
    for (let key in data) {
      result = result.replace(new RegExp(`\\
    }
    return result;
  }
}
	template

In pure JS	

function throttle(func, wait) {
  let timeout;
  return function (...args) {
    const context = this;
    if (!timeout) {
      func.apply(context, args);
      timeout = setTimeout(() => {
        timeout = null;
      }, wait);
    }
  };
}
	throttle

function times(n, iteratee) {
  const result = Array(n);
  for (let i = 0; i < n; i++) {
    result[i] = iteratee(i);
  }
  return result;
}
	times

function toArray(value) {
  if (value == null) {
    return [];
  }
  if (Array.isArray(value)) {
    return value.slice();
  }
  if (typeof value === 'object') {
    return Object.values(value);
  }
  return [value];
}
	toArray

function toPath(path) {
  if (Array.isArray(path)) {
    return path;
  }
  if (typeof path === 'string') {
    return path.split(/[.[\]]/).filter(Boolean);
  }
  return [path];
}
	toPath

function transform(collection, iteratee, acc
  // check if accumulator is undefined, if it
  accumulator = accumulator !== undefine
 	
  // iterate over the collection
  for (let i = 0; i < collection.length; i++) {
    accumulator = iteratee(accumulator, co
  }
 	
  return accumulator;
}
	transform

function trim(str) {
  return str.replace(/^\s+|\s+$/g, '');
}
	trim

In pure JS	

function trimEnd(string, chars = ' ') {
  if (string && chars) {
    const regex = new RegExp(`[${chars}]`
    return string.replace(regex, '').replace(/
  }
  return string.trim();
}
	trimEnd

if (!String.prototype.trimStart) {
  String.prototype.trimStart = String.protot
    return this.replace(/^\s+/, '');
  };
}
	trimStart

function truncate(str, options) {
  if (str.length <= options.length) {
    return str;
  }
  const separator = options.omission || '...'
  const charsToShow = options.length - se
  const truncatedString = str.slice(0, chars
  return truncatedString + separator;
}
	truncate

function union(...arrays) {
  return [...new Set(arrays.flat())];
}
	union

function uniq(arr) {
  return Array.from(new Set(arr));
}
	uniq

function uniqBy(array, iteratee) {
  const seen = new Set();
  return array.filter((element) => {
    const key = iteratee(element);
    if (seen.has(key)) {
      return false;
    } else {
      seen.add(key);
      return true;
    }
  });
}
	uniqBy

let id = 0;
 	
function uniqueId(prefix = '') {
  id += 1;
  return prefix + id;
}
	uniqueId

function unzip(arr) {
  const maxLength = Math.max(...arr.map
  const result = new Array(maxLength);
 	
  for (let i = 0; i < maxLength; i++) {
    result[i] = arr.map(a => a[i]);
  }
 	
  return result;
}
	unzip

In pure JS	

function upperFirst(str) {
  if (typeof str !== 'string' || str.length ===
    return '';
  }
  return str.charAt(0).toUpperCase() + str.
}
	upperFirst

Object.values	values

function where(array, properties) {
  return array.filter(function(obj) {
    for (var key in properties) {
      if (obj[key] !== properties[key]) {
        return false;
      }
    }
    return true;
  });
}
	where

function without(array, ...values) {
  return array.filter(item => !values.include
}
	without

function words(str, pattern) {
  pattern = pattern || /\w+/g;
  return str.match(pattern);
}
	words

function wrap(func, wrapper) {
  return function(...args) {
    return wrapper(func, ...args);
  };
}
	wrap

function zip(...arrays) {
  const maxLength = Math.max(...arrays.
  const result = [];
 	
  for (let i = 0; i < maxLength; i++) {
    result.push(arrays.map(a => a[i]));
  }
 	
  return result;
}
	zip

function zipObject(props, values) {
  return props.reduce((obj, prop, index) =
    obj[prop] = values[index];
    return obj;
  }, {});
}
	zipObject