/\*! \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) Microsoft Corporation. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use

this file except in compliance with the License. You may obtain a copy of the

License at http://www.apache.org/licenses/LICENSE-2.0

THIS CODE IS PROVIDED ON AN \*AS IS\* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED

WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,

MERCHANTABLITY OR NON-INFRINGEMENT.

See the Apache Version 2.0 License for specific language governing permissions

and limitations under the License.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/// <reference no-default-lib="true"/>

/////////////////////////////

/// ECMAScript APIs

/////////////////////////////

declare var NaN: number;

declare var Infinity: number;

/\*\*

\* Evaluates JavaScript code and executes it.

\* @param x A String value that contains valid JavaScript code.

\*/

declare function eval(x: string): any;

/\*\*

\* Converts a string to an integer.

\* @param s A string to convert into a number.

\* @param radix A value between 2 and 36 that specifies the base of the number in numString.

\* If this argument is not supplied, strings with a prefix of '0x' are considered hexadecimal.

\* All other strings are considered decimal.

\*/

declare function parseInt(s: string, radix?: number): number;

/\*\*

\* Converts a string to a floating-point number.

\* @param string A string that contains a floating-point number.

\*/

declare function parseFloat(string: string): number;

/\*\*

\* Returns a Boolean value that indicates whether a value is the reserved value NaN (not a number).

\* @param number A numeric value.

\*/

declare function isNaN(number: number): boolean;

/\*\*

\* Determines whether a supplied number is finite.

\* @param number Any numeric value.

\*/

declare function isFinite(number: number): boolean;

/\*\*

\* Gets the unencoded version of an encoded Uniform Resource Identifier (URI).

\* @param encodedURI A value representing an encoded URI.

\*/

declare function decodeURI(encodedURI: string): string;

/\*\*

\* Gets the unencoded version of an encoded component of a Uniform Resource Identifier (URI).

\* @param encodedURIComponent A value representing an encoded URI component.

\*/

declare function decodeURIComponent(encodedURIComponent: string): string;

/\*\*

\* Encodes a text string as a valid Uniform Resource Identifier (URI)

\* @param uri A value representing an encoded URI.

\*/

declare function encodeURI(uri: string): string;

/\*\*

\* Encodes a text string as a valid component of a Uniform Resource Identifier (URI).

\* @param uriComponent A value representing an encoded URI component.

\*/

declare function encodeURIComponent(uriComponent: string | number | boolean): string;

/\*\*

\* Computes a new string in which certain characters have been replaced by a hexadecimal escape sequence.

\* @param string A string value

\*/

declare function escape(string: string): string;

/\*\*

\* Computes a new string in which hexadecimal escape sequences are replaced with the character that it represents.

\* @param string A string value

\*/

declare function unescape(string: string): string;

interface Symbol {

/\*\* Returns a string representation of an object. \*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): symbol;

}

declare type PropertyKey = string | number | symbol;

interface PropertyDescriptor {

configurable?: boolean;

enumerable?: boolean;

value?: any;

writable?: boolean;

get?(): any;

set?(v: any): void;

}

interface PropertyDescriptorMap {

[s: string]: PropertyDescriptor;

}

interface Object {

/\*\* The initial value of Object.prototype.constructor is the standard built-in Object constructor. \*/

constructor: Function;

/\*\* Returns a string representation of an object. \*/

toString(): string;

/\*\* Returns a date converted to a string using the current locale. \*/

toLocaleString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Object;

/\*\*

\* Determines whether an object has a property with the specified name.

\* @param v A property name.

\*/

hasOwnProperty(v: PropertyKey): boolean;

/\*\*

\* Determines whether an object exists in another object's prototype chain.

\* @param v Another object whose prototype chain is to be checked.

\*/

isPrototypeOf(v: Object): boolean;

/\*\*

\* Determines whether a specified property is enumerable.

\* @param v A property name.

\*/

propertyIsEnumerable(v: PropertyKey): boolean;

}

interface ObjectConstructor {

new(value?: any): Object;

(): any;

(value: any): any;

/\*\* A reference to the prototype for a class of objects. \*/

readonly prototype: Object;

/\*\*

\* Returns the prototype of an object.

\* @param o The object that references the prototype.

\*/

getPrototypeOf(o: any): any;

/\*\*

\* Gets the own property descriptor of the specified object.

\* An own property descriptor is one that is defined directly on the object and is not inherited from the object's prototype.

\* @param o Object that contains the property.

\* @param p Name of the property.

\*/

getOwnPropertyDescriptor(o: any, p: PropertyKey): PropertyDescriptor | undefined;

/\*\*

\* Returns the names of the own properties of an object. The own properties of an object are those that are defined directly

\* on that object, and are not inherited from the object's prototype. The properties of an object include both fields (objects) and functions.

\* @param o Object that contains the own properties.

\*/

getOwnPropertyNames(o: any): string[];

/\*\*

\* Creates an object that has the specified prototype or that has null prototype.

\* @param o Object to use as a prototype. May be null.

\*/

create(o: object | null): any;

/\*\*

\* Creates an object that has the specified prototype, and that optionally contains specified properties.

\* @param o Object to use as a prototype. May be null

\* @param properties JavaScript object that contains one or more property descriptors.

\*/

create(o: object | null, properties: PropertyDescriptorMap & ThisType<any>): any;

/\*\*

\* Adds a property to an object, or modifies attributes of an existing property.

\* @param o Object on which to add or modify the property. This can be a native JavaScript object (that is, a user-defined object or a built in object) or a DOM object.

\* @param p The property name.

\* @param attributes Descriptor for the property. It can be for a data property or an accessor property.

\*/

defineProperty(o: any, p: PropertyKey, attributes: PropertyDescriptor & ThisType<any>): any;

/\*\*

\* Adds one or more properties to an object, and/or modifies attributes of existing properties.

\* @param o Object on which to add or modify the properties. This can be a native JavaScript object or a DOM object.

\* @param properties JavaScript object that contains one or more descriptor objects. Each descriptor object describes a data property or an accessor property.

\*/

defineProperties(o: any, properties: PropertyDescriptorMap & ThisType<any>): any;

/\*\*

\* Prevents the modification of attributes of existing properties, and prevents the addition of new properties.

\* @param o Object on which to lock the attributes.

\*/

seal<T>(o: T): T;

/\*\*

\* Prevents the modification of existing property attributes and values, and prevents the addition of new properties.

\* @param o Object on which to lock the attributes.

\*/

freeze<T>(a: T[]): readonly T[];

/\*\*

\* Prevents the modification of existing property attributes and values, and prevents the addition of new properties.

\* @param o Object on which to lock the attributes.

\*/

freeze<T extends Function>(f: T): T;

/\*\*

\* Prevents the modification of existing property attributes and values, and prevents the addition of new properties.

\* @param o Object on which to lock the attributes.

\*/

freeze<T>(o: T): Readonly<T>;

/\*\*

\* Prevents the addition of new properties to an object.

\* @param o Object to make non-extensible.

\*/

preventExtensions<T>(o: T): T;

/\*\*

\* Returns true if existing property attributes cannot be modified in an object and new properties cannot be added to the object.

\* @param o Object to test.

\*/

isSealed(o: any): boolean;

/\*\*

\* Returns true if existing property attributes and values cannot be modified in an object, and new properties cannot be added to the object.

\* @param o Object to test.

\*/

isFrozen(o: any): boolean;

/\*\*

\* Returns a value that indicates whether new properties can be added to an object.

\* @param o Object to test.

\*/

isExtensible(o: any): boolean;

/\*\*

\* Returns the names of the enumerable string properties and methods of an object.

\* @param o Object that contains the properties and methods. This can be an object that you created or an existing Document Object Model (DOM) object.

\*/

keys(o: object): string[];

}

/\*\*

\* Provides functionality common to all JavaScript objects.

\*/

declare var Object: ObjectConstructor;

/\*\*

\* Creates a new function.

\*/

interface Function {

/\*\*

\* Calls the function, substituting the specified object for the this value of the function, and the specified array for the arguments of the function.

\* @param thisArg The object to be used as the this object.

\* @param argArray A set of arguments to be passed to the function.

\*/

apply(this: Function, thisArg: any, argArray?: any): any;

/\*\*

\* Calls a method of an object, substituting another object for the current object.

\* @param thisArg The object to be used as the current object.

\* @param argArray A list of arguments to be passed to the method.

\*/

call(this: Function, thisArg: any, ...argArray: any[]): any;

/\*\*

\* For a given function, creates a bound function that has the same body as the original function.

\* The this object of the bound function is associated with the specified object, and has the specified initial parameters.

\* @param thisArg An object to which the this keyword can refer inside the new function.

\* @param argArray A list of arguments to be passed to the new function.

\*/

bind(this: Function, thisArg: any, ...argArray: any[]): any;

/\*\* Returns a string representation of a function. \*/

toString(): string;

prototype: any;

readonly length: number;

// Non-standard extensions

arguments: any;

caller: Function;

}

interface FunctionConstructor {

/\*\*

\* Creates a new function.

\* @param args A list of arguments the function accepts.

\*/

new(...args: string[]): Function;

(...args: string[]): Function;

readonly prototype: Function;

}

declare var Function: FunctionConstructor;

/\*\*

\* Extracts the type of the 'this' parameter of a function type, or 'unknown' if the function type has no 'this' parameter.

\*/

type ThisParameterType<T> = T extends (this: infer U, ...args: any[]) => any ? U : unknown;

/\*\*

\* Removes the 'this' parameter from a function type.

\*/

type OmitThisParameter<T> = unknown extends ThisParameterType<T> ? T : T extends (...args: infer A) => infer R ? (...args: A) => R : T;

interface CallableFunction extends Function {

/\*\*

\* Calls the function with the specified object as the this value and the elements of specified array as the arguments.

\* @param thisArg The object to be used as the this object.

\* @param args An array of argument values to be passed to the function.

\*/

apply<T, R>(this: (this: T) => R, thisArg: T): R;

apply<T, A extends any[], R>(this: (this: T, ...args: A) => R, thisArg: T, args: A): R;

/\*\*

\* Calls the function with the specified object as the this value and the specified rest arguments as the arguments.

\* @param thisArg The object to be used as the this object.

\* @param args Argument values to be passed to the function.

\*/

call<T, A extends any[], R>(this: (this: T, ...args: A) => R, thisArg: T, ...args: A): R;

/\*\*

\* For a given function, creates a bound function that has the same body as the original function.

\* The this object of the bound function is associated with the specified object, and has the specified initial parameters.

\* @param thisArg The object to be used as the this object.

\* @param args Arguments to bind to the parameters of the function.

\*/

bind<T>(this: T, thisArg: ThisParameterType<T>): OmitThisParameter<T>;

bind<T, A0, A extends any[], R>(this: (this: T, arg0: A0, ...args: A) => R, thisArg: T, arg0: A0): (...args: A) => R;

bind<T, A0, A1, A extends any[], R>(this: (this: T, arg0: A0, arg1: A1, ...args: A) => R, thisArg: T, arg0: A0, arg1: A1): (...args: A) => R;

bind<T, A0, A1, A2, A extends any[], R>(this: (this: T, arg0: A0, arg1: A1, arg2: A2, ...args: A) => R, thisArg: T, arg0: A0, arg1: A1, arg2: A2): (...args: A) => R;

bind<T, A0, A1, A2, A3, A extends any[], R>(this: (this: T, arg0: A0, arg1: A1, arg2: A2, arg3: A3, ...args: A) => R, thisArg: T, arg0: A0, arg1: A1, arg2: A2, arg3: A3): (...args: A) => R;

bind<T, AX, R>(this: (this: T, ...args: AX[]) => R, thisArg: T, ...args: AX[]): (...args: AX[]) => R;

}

interface NewableFunction extends Function {

/\*\*

\* Calls the function with the specified object as the this value and the elements of specified array as the arguments.

\* @param thisArg The object to be used as the this object.

\* @param args An array of argument values to be passed to the function.

\*/

apply<T>(this: new () => T, thisArg: T): void;

apply<T, A extends any[]>(this: new (...args: A) => T, thisArg: T, args: A): void;

/\*\*

\* Calls the function with the specified object as the this value and the specified rest arguments as the arguments.

\* @param thisArg The object to be used as the this object.

\* @param args Argument values to be passed to the function.

\*/

call<T, A extends any[]>(this: new (...args: A) => T, thisArg: T, ...args: A): void;

/\*\*

\* For a given function, creates a bound function that has the same body as the original function.

\* The this object of the bound function is associated with the specified object, and has the specified initial parameters.

\* @param thisArg The object to be used as the this object.

\* @param args Arguments to bind to the parameters of the function.

\*/

bind<T>(this: T, thisArg: any): T;

bind<A0, A extends any[], R>(this: new (arg0: A0, ...args: A) => R, thisArg: any, arg0: A0): new (...args: A) => R;

bind<A0, A1, A extends any[], R>(this: new (arg0: A0, arg1: A1, ...args: A) => R, thisArg: any, arg0: A0, arg1: A1): new (...args: A) => R;

bind<A0, A1, A2, A extends any[], R>(this: new (arg0: A0, arg1: A1, arg2: A2, ...args: A) => R, thisArg: any, arg0: A0, arg1: A1, arg2: A2): new (...args: A) => R;

bind<A0, A1, A2, A3, A extends any[], R>(this: new (arg0: A0, arg1: A1, arg2: A2, arg3: A3, ...args: A) => R, thisArg: any, arg0: A0, arg1: A1, arg2: A2, arg3: A3): new (...args: A) => R;

bind<AX, R>(this: new (...args: AX[]) => R, thisArg: any, ...args: AX[]): new (...args: AX[]) => R;

}

interface IArguments {

[index: number]: any;

length: number;

callee: Function;

}

interface String {

/\*\* Returns a string representation of a string. \*/

toString(): string;

/\*\*

\* Returns the character at the specified index.

\* @param pos The zero-based index of the desired character.

\*/

charAt(pos: number): string;

/\*\*

\* Returns the Unicode value of the character at the specified location.

\* @param index The zero-based index of the desired character. If there is no character at the specified index, NaN is returned.

\*/

charCodeAt(index: number): number;

/\*\*

\* Returns a string that contains the concatenation of two or more strings.

\* @param strings The strings to append to the end of the string.

\*/

concat(...strings: string[]): string;

/\*\*

\* Returns the position of the first occurrence of a substring.

\* @param searchString The substring to search for in the string

\* @param position The index at which to begin searching the String object. If omitted, search starts at the beginning of the string.

\*/

indexOf(searchString: string, position?: number): number;

/\*\*

\* Returns the last occurrence of a substring in the string.

\* @param searchString The substring to search for.

\* @param position The index at which to begin searching. If omitted, the search begins at the end of the string.

\*/

lastIndexOf(searchString: string, position?: number): number;

/\*\*

\* Determines whether two strings are equivalent in the current locale.

\* @param that String to compare to target string

\*/

localeCompare(that: string): number;

/\*\*

\* Matches a string with a regular expression, and returns an array containing the results of that search.

\* @param regexp A variable name or string literal containing the regular expression pattern and flags.

\*/

match(regexp: string | RegExp): RegExpMatchArray | null;

/\*\*

\* Replaces text in a string, using a regular expression or search string.

\* @param searchValue A string to search for.

\* @param replaceValue A string containing the text to replace for every successful match of searchValue in this string.

\*/

replace(searchValue: string | RegExp, replaceValue: string): string;

/\*\*

\* Replaces text in a string, using a regular expression or search string.

\* @param searchValue A string to search for.

\* @param replacer A function that returns the replacement text.

\*/

replace(searchValue: string | RegExp, replacer: (substring: string, ...args: any[]) => string): string;

/\*\*

\* Finds the first substring match in a regular expression search.

\* @param regexp The regular expression pattern and applicable flags.

\*/

search(regexp: string | RegExp): number;

/\*\*

\* Returns a section of a string.

\* @param start The index to the beginning of the specified portion of stringObj.

\* @param end The index to the end of the specified portion of stringObj. The substring includes the characters up to, but not including, the character indicated by end.

\* If this value is not specified, the substring continues to the end of stringObj.

\*/

slice(start?: number, end?: number): string;

/\*\*

\* Split a string into substrings using the specified separator and return them as an array.

\* @param separator A string that identifies character or characters to use in separating the string. If omitted, a single-element array containing the entire string is returned.

\* @param limit A value used to limit the number of elements returned in the array.

\*/

split(separator: string | RegExp, limit?: number): string[];

/\*\*

\* Returns the substring at the specified location within a String object.

\* @param start The zero-based index number indicating the beginning of the substring.

\* @param end Zero-based index number indicating the end of the substring. The substring includes the characters up to, but not including, the character indicated by end.

\* If end is omitted, the characters from start through the end of the original string are returned.

\*/

substring(start: number, end?: number): string;

/\*\* Converts all the alphabetic characters in a string to lowercase. \*/

toLowerCase(): string;

/\*\* Converts all alphabetic characters to lowercase, taking into account the host environment's current locale. \*/

toLocaleLowerCase(locales?: string | string[]): string;

/\*\* Converts all the alphabetic characters in a string to uppercase. \*/

toUpperCase(): string;

/\*\* Returns a string where all alphabetic characters have been converted to uppercase, taking into account the host environment's current locale. \*/

toLocaleUpperCase(locales?: string | string[]): string;

/\*\* Removes the leading and trailing white space and line terminator characters from a string. \*/

trim(): string;

/\*\* Returns the length of a String object. \*/

readonly length: number;

// IE extensions

/\*\*

\* Gets a substring beginning at the specified location and having the specified length.

\* @param from The starting position of the desired substring. The index of the first character in the string is zero.

\* @param length The number of characters to include in the returned substring.

\*/

substr(from: number, length?: number): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): string;

readonly [index: number]: string;

}

interface StringConstructor {

new(value?: any): String;

(value?: any): string;

readonly prototype: String;

fromCharCode(...codes: number[]): string;

}

/\*\*

\* Allows manipulation and formatting of text strings and determination and location of substrings within strings.

\*/

declare var String: StringConstructor;

interface Boolean {

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): boolean;

}

interface BooleanConstructor {

new(value?: any): Boolean;

<T>(value?: T): boolean;

readonly prototype: Boolean;

}

declare var Boolean: BooleanConstructor;

interface Number {

/\*\*

\* Returns a string representation of an object.

\* @param radix Specifies a radix for converting numeric values to strings. This value is only used for numbers.

\*/

toString(radix?: number): string;

/\*\*

\* Returns a string representing a number in fixed-point notation.

\* @param fractionDigits Number of digits after the decimal point. Must be in the range 0 - 20, inclusive.

\*/

toFixed(fractionDigits?: number): string;

/\*\*

\* Returns a string containing a number represented in exponential notation.

\* @param fractionDigits Number of digits after the decimal point. Must be in the range 0 - 20, inclusive.

\*/

toExponential(fractionDigits?: number): string;

/\*\*

\* Returns a string containing a number represented either in exponential or fixed-point notation with a specified number of digits.

\* @param precision Number of significant digits. Must be in the range 1 - 21, inclusive.

\*/

toPrecision(precision?: number): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): number;

}

interface NumberConstructor {

new(value?: any): Number;

(value?: any): number;

readonly prototype: Number;

/\*\* The largest number that can be represented in JavaScript. Equal to approximately 1.79E+308. \*/

readonly MAX\_VALUE: number;

/\*\* The closest number to zero that can be represented in JavaScript. Equal to approximately 5.00E-324. \*/

readonly MIN\_VALUE: number;

/\*\*

\* A value that is not a number.

\* In equality comparisons, NaN does not equal any value, including itself. To test whether a value is equivalent to NaN, use the isNaN function.

\*/

readonly NaN: number;

/\*\*

\* A value that is less than the largest negative number that can be represented in JavaScript.

\* JavaScript displays NEGATIVE\_INFINITY values as -infinity.

\*/

readonly NEGATIVE\_INFINITY: number;

/\*\*

\* A value greater than the largest number that can be represented in JavaScript.

\* JavaScript displays POSITIVE\_INFINITY values as infinity.

\*/

readonly POSITIVE\_INFINITY: number;

}

/\*\* An object that represents a number of any kind. All JavaScript numbers are 64-bit floating-point numbers. \*/

declare var Number: NumberConstructor;

interface TemplateStringsArray extends ReadonlyArray<string> {

readonly raw: readonly string[];

}

/\*\*

\* The type of `import.meta`.

\*

\* If you need to declare that a given property exists on `import.meta`,

\* this type may be augmented via interface merging.

\*/

interface ImportMeta {

}

interface Math {

/\*\* The mathematical constant e. This is Euler's number, the base of natural logarithms. \*/

readonly E: number;

/\*\* The natural logarithm of 10. \*/

readonly LN10: number;

/\*\* The natural logarithm of 2. \*/

readonly LN2: number;

/\*\* The base-2 logarithm of e. \*/

readonly LOG2E: number;

/\*\* The base-10 logarithm of e. \*/

readonly LOG10E: number;

/\*\* Pi. This is the ratio of the circumference of a circle to its diameter. \*/

readonly PI: number;

/\*\* The square root of 0.5, or, equivalently, one divided by the square root of 2. \*/

readonly SQRT1\_2: number;

/\*\* The square root of 2. \*/

readonly SQRT2: number;

/\*\*

\* Returns the absolute value of a number (the value without regard to whether it is positive or negative).

\* For example, the absolute value of -5 is the same as the absolute value of 5.

\* @param x A numeric expression for which the absolute value is needed.

\*/

abs(x: number): number;

/\*\*

\* Returns the arc cosine (or inverse cosine) of a number.

\* @param x A numeric expression.

\*/

acos(x: number): number;

/\*\*

\* Returns the arcsine of a number.

\* @param x A numeric expression.

\*/

asin(x: number): number;

/\*\*

\* Returns the arctangent of a number.

\* @param x A numeric expression for which the arctangent is needed.

\*/

atan(x: number): number;

/\*\*

\* Returns the angle (in radians) from the X axis to a point.

\* @param y A numeric expression representing the cartesian y-coordinate.

\* @param x A numeric expression representing the cartesian x-coordinate.

\*/

atan2(y: number, x: number): number;

/\*\*

\* Returns the smallest integer greater than or equal to its numeric argument.

\* @param x A numeric expression.

\*/

ceil(x: number): number;

/\*\*

\* Returns the cosine of a number.

\* @param x A numeric expression that contains an angle measured in radians.

\*/

cos(x: number): number;

/\*\*

\* Returns e (the base of natural logarithms) raised to a power.

\* @param x A numeric expression representing the power of e.

\*/

exp(x: number): number;

/\*\*

\* Returns the greatest integer less than or equal to its numeric argument.

\* @param x A numeric expression.

\*/

floor(x: number): number;

/\*\*

\* Returns the natural logarithm (base e) of a number.

\* @param x A numeric expression.

\*/

log(x: number): number;

/\*\*

\* Returns the larger of a set of supplied numeric expressions.

\* @param values Numeric expressions to be evaluated.

\*/

max(...values: number[]): number;

/\*\*

\* Returns the smaller of a set of supplied numeric expressions.

\* @param values Numeric expressions to be evaluated.

\*/

min(...values: number[]): number;

/\*\*

\* Returns the value of a base expression taken to a specified power.

\* @param x The base value of the expression.

\* @param y The exponent value of the expression.

\*/

pow(x: number, y: number): number;

/\*\* Returns a pseudorandom number between 0 and 1. \*/

random(): number;

/\*\*

\* Returns a supplied numeric expression rounded to the nearest integer.

\* @param x The value to be rounded to the nearest integer.

\*/

round(x: number): number;

/\*\*

\* Returns the sine of a number.

\* @param x A numeric expression that contains an angle measured in radians.

\*/

sin(x: number): number;

/\*\*

\* Returns the square root of a number.

\* @param x A numeric expression.

\*/

sqrt(x: number): number;

/\*\*

\* Returns the tangent of a number.

\* @param x A numeric expression that contains an angle measured in radians.

\*/

tan(x: number): number;

}

/\*\* An intrinsic object that provides basic mathematics functionality and constants. \*/

declare var Math: Math;

/\*\* Enables basic storage and retrieval of dates and times. \*/

interface Date {

/\*\* Returns a string representation of a date. The format of the string depends on the locale. \*/

toString(): string;

/\*\* Returns a date as a string value. \*/

toDateString(): string;

/\*\* Returns a time as a string value. \*/

toTimeString(): string;

/\*\* Returns a value as a string value appropriate to the host environment's current locale. \*/

toLocaleString(): string;

/\*\* Returns a date as a string value appropriate to the host environment's current locale. \*/

toLocaleDateString(): string;

/\*\* Returns a time as a string value appropriate to the host environment's current locale. \*/

toLocaleTimeString(): string;

/\*\* Returns the stored time value in milliseconds since midnight, January 1, 1970 UTC. \*/

valueOf(): number;

/\*\* Gets the time value in milliseconds. \*/

getTime(): number;

/\*\* Gets the year, using local time. \*/

getFullYear(): number;

/\*\* Gets the year using Universal Coordinated Time (UTC). \*/

getUTCFullYear(): number;

/\*\* Gets the month, using local time. \*/

getMonth(): number;

/\*\* Gets the month of a Date object using Universal Coordinated Time (UTC). \*/

getUTCMonth(): number;

/\*\* Gets the day-of-the-month, using local time. \*/

getDate(): number;

/\*\* Gets the day-of-the-month, using Universal Coordinated Time (UTC). \*/

getUTCDate(): number;

/\*\* Gets the day of the week, using local time. \*/

getDay(): number;

/\*\* Gets the day of the week using Universal Coordinated Time (UTC). \*/

getUTCDay(): number;

/\*\* Gets the hours in a date, using local time. \*/

getHours(): number;

/\*\* Gets the hours value in a Date object using Universal Coordinated Time (UTC). \*/

getUTCHours(): number;

/\*\* Gets the minutes of a Date object, using local time. \*/

getMinutes(): number;

/\*\* Gets the minutes of a Date object using Universal Coordinated Time (UTC). \*/

getUTCMinutes(): number;

/\*\* Gets the seconds of a Date object, using local time. \*/

getSeconds(): number;

/\*\* Gets the seconds of a Date object using Universal Coordinated Time (UTC). \*/

getUTCSeconds(): number;

/\*\* Gets the milliseconds of a Date, using local time. \*/

getMilliseconds(): number;

/\*\* Gets the milliseconds of a Date object using Universal Coordinated Time (UTC). \*/

getUTCMilliseconds(): number;

/\*\* Gets the difference in minutes between the time on the local computer and Universal Coordinated Time (UTC). \*/

getTimezoneOffset(): number;

/\*\*

\* Sets the date and time value in the Date object.

\* @param time A numeric value representing the number of elapsed milliseconds since midnight, January 1, 1970 GMT.

\*/

setTime(time: number): number;

/\*\*

\* Sets the milliseconds value in the Date object using local time.

\* @param ms A numeric value equal to the millisecond value.

\*/

setMilliseconds(ms: number): number;

/\*\*

\* Sets the milliseconds value in the Date object using Universal Coordinated Time (UTC).

\* @param ms A numeric value equal to the millisecond value.

\*/

setUTCMilliseconds(ms: number): number;

/\*\*

\* Sets the seconds value in the Date object using local time.

\* @param sec A numeric value equal to the seconds value.

\* @param ms A numeric value equal to the milliseconds value.

\*/

setSeconds(sec: number, ms?: number): number;

/\*\*

\* Sets the seconds value in the Date object using Universal Coordinated Time (UTC).

\* @param sec A numeric value equal to the seconds value.

\* @param ms A numeric value equal to the milliseconds value.

\*/

setUTCSeconds(sec: number, ms?: number): number;

/\*\*

\* Sets the minutes value in the Date object using local time.

\* @param min A numeric value equal to the minutes value.

\* @param sec A numeric value equal to the seconds value.

\* @param ms A numeric value equal to the milliseconds value.

\*/

setMinutes(min: number, sec?: number, ms?: number): number;

/\*\*

\* Sets the minutes value in the Date object using Universal Coordinated Time (UTC).

\* @param min A numeric value equal to the minutes value.

\* @param sec A numeric value equal to the seconds value.

\* @param ms A numeric value equal to the milliseconds value.

\*/

setUTCMinutes(min: number, sec?: number, ms?: number): number;

/\*\*

\* Sets the hour value in the Date object using local time.

\* @param hours A numeric value equal to the hours value.

\* @param min A numeric value equal to the minutes value.

\* @param sec A numeric value equal to the seconds value.

\* @param ms A numeric value equal to the milliseconds value.

\*/

setHours(hours: number, min?: number, sec?: number, ms?: number): number;

/\*\*

\* Sets the hours value in the Date object using Universal Coordinated Time (UTC).

\* @param hours A numeric value equal to the hours value.

\* @param min A numeric value equal to the minutes value.

\* @param sec A numeric value equal to the seconds value.

\* @param ms A numeric value equal to the milliseconds value.

\*/

setUTCHours(hours: number, min?: number, sec?: number, ms?: number): number;

/\*\*

\* Sets the numeric day-of-the-month value of the Date object using local time.

\* @param date A numeric value equal to the day of the month.

\*/

setDate(date: number): number;

/\*\*

\* Sets the numeric day of the month in the Date object using Universal Coordinated Time (UTC).

\* @param date A numeric value equal to the day of the month.

\*/

setUTCDate(date: number): number;

/\*\*

\* Sets the month value in the Date object using local time.

\* @param month A numeric value equal to the month. The value for January is 0, and other month values follow consecutively.

\* @param date A numeric value representing the day of the month. If this value is not supplied, the value from a call to the getDate method is used.

\*/

setMonth(month: number, date?: number): number;

/\*\*

\* Sets the month value in the Date object using Universal Coordinated Time (UTC).

\* @param month A numeric value equal to the month. The value for January is 0, and other month values follow consecutively.

\* @param date A numeric value representing the day of the month. If it is not supplied, the value from a call to the getUTCDate method is used.

\*/

setUTCMonth(month: number, date?: number): number;

/\*\*

\* Sets the year of the Date object using local time.

\* @param year A numeric value for the year.

\* @param month A zero-based numeric value for the month (0 for January, 11 for December). Must be specified if numDate is specified.

\* @param date A numeric value equal for the day of the month.

\*/

setFullYear(year: number, month?: number, date?: number): number;

/\*\*

\* Sets the year value in the Date object using Universal Coordinated Time (UTC).

\* @param year A numeric value equal to the year.

\* @param month A numeric value equal to the month. The value for January is 0, and other month values follow consecutively. Must be supplied if numDate is supplied.

\* @param date A numeric value equal to the day of the month.

\*/

setUTCFullYear(year: number, month?: number, date?: number): number;

/\*\* Returns a date converted to a string using Universal Coordinated Time (UTC). \*/

toUTCString(): string;

/\*\* Returns a date as a string value in ISO format. \*/

toISOString(): string;

/\*\* Used by the JSON.stringify method to enable the transformation of an object's data for JavaScript Object Notation (JSON) serialization. \*/

toJSON(key?: any): string;

}

interface DateConstructor {

new(): Date;

new(value: number | string): Date;

new(year: number, month: number, date?: number, hours?: number, minutes?: number, seconds?: number, ms?: number): Date;

(): string;

readonly prototype: Date;

/\*\*

\* Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.

\* @param s A date string

\*/

parse(s: string): number;

/\*\*

\* Returns the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (UTC) (or GMT) and the specified date.

\* @param year The full year designation is required for cross-century date accuracy. If year is between 0 and 99 is used, then year is assumed to be 1900 + year.

\* @param month The month as a number between 0 and 11 (January to December).

\* @param date The date as a number between 1 and 31.

\* @param hours Must be supplied if minutes is supplied. A number from 0 to 23 (midnight to 11pm) that specifies the hour.

\* @param minutes Must be supplied if seconds is supplied. A number from 0 to 59 that specifies the minutes.

\* @param seconds Must be supplied if milliseconds is supplied. A number from 0 to 59 that specifies the seconds.

\* @param ms A number from 0 to 999 that specifies the milliseconds.

\*/

UTC(year: number, month: number, date?: number, hours?: number, minutes?: number, seconds?: number, ms?: number): number;

now(): number;

}

declare var Date: DateConstructor;

interface RegExpMatchArray extends Array<string> {

index?: number;

input?: string;

}

interface RegExpExecArray extends Array<string> {

index: number;

input: string;

}

interface RegExp {

/\*\*

\* Executes a search on a string using a regular expression pattern, and returns an array containing the results of that search.

\* @param string The String object or string literal on which to perform the search.

\*/

exec(string: string): RegExpExecArray | null;

/\*\*

\* Returns a Boolean value that indicates whether or not a pattern exists in a searched string.

\* @param string String on which to perform the search.

\*/

test(string: string): boolean;

/\*\* Returns a copy of the text of the regular expression pattern. Read-only. The regExp argument is a Regular expression object. It can be a variable name or a literal. \*/

readonly source: string;

/\*\* Returns a Boolean value indicating the state of the global flag (g) used with a regular expression. Default is false. Read-only. \*/

readonly global: boolean;

/\*\* Returns a Boolean value indicating the state of the ignoreCase flag (i) used with a regular expression. Default is false. Read-only. \*/

readonly ignoreCase: boolean;

/\*\* Returns a Boolean value indicating the state of the multiline flag (m) used with a regular expression. Default is false. Read-only. \*/

readonly multiline: boolean;

lastIndex: number;

// Non-standard extensions

compile(): this;

}

interface RegExpConstructor {

new(pattern: RegExp | string): RegExp;

new(pattern: string, flags?: string): RegExp;

(pattern: RegExp | string): RegExp;

(pattern: string, flags?: string): RegExp;

readonly prototype: RegExp;

// Non-standard extensions

$1: string;

$2: string;

$3: string;

$4: string;

$5: string;

$6: string;

$7: string;

$8: string;

$9: string;

lastMatch: string;

}

declare var RegExp: RegExpConstructor;

interface Error {

name: string;

message: string;

stack?: string;

}

interface ErrorConstructor {

new(message?: string): Error;

(message?: string): Error;

readonly prototype: Error;

}

declare var Error: ErrorConstructor;

interface EvalError extends Error {

}

interface EvalErrorConstructor extends ErrorConstructor {

new(message?: string): EvalError;

(message?: string): EvalError;

readonly prototype: EvalError;

}

declare var EvalError: EvalErrorConstructor;

interface RangeError extends Error {

}

interface RangeErrorConstructor extends ErrorConstructor {

new(message?: string): RangeError;

(message?: string): RangeError;

readonly prototype: RangeError;

}

declare var RangeError: RangeErrorConstructor;

interface ReferenceError extends Error {

}

interface ReferenceErrorConstructor extends ErrorConstructor {

new(message?: string): ReferenceError;

(message?: string): ReferenceError;

readonly prototype: ReferenceError;

}

declare var ReferenceError: ReferenceErrorConstructor;

interface SyntaxError extends Error {

}

interface SyntaxErrorConstructor extends ErrorConstructor {

new(message?: string): SyntaxError;

(message?: string): SyntaxError;

readonly prototype: SyntaxError;

}

declare var SyntaxError: SyntaxErrorConstructor;

interface TypeError extends Error {

}

interface TypeErrorConstructor extends ErrorConstructor {

new(message?: string): TypeError;

(message?: string): TypeError;

readonly prototype: TypeError;

}

declare var TypeError: TypeErrorConstructor;

interface URIError extends Error {

}

interface URIErrorConstructor extends ErrorConstructor {

new(message?: string): URIError;

(message?: string): URIError;

readonly prototype: URIError;

}

declare var URIError: URIErrorConstructor;

interface JSON {

/\*\*

\* Converts a JavaScript Object Notation (JSON) string into an object.

\* @param text A valid JSON string.

\* @param reviver A function that transforms the results. This function is called for each member of the object.

\* If a member contains nested objects, the nested objects are transformed before the parent object is.

\*/

parse(text: string, reviver?: (this: any, key: string, value: any) => any): any;

/\*\*

\* Converts a JavaScript value to a JavaScript Object Notation (JSON) string.

\* @param value A JavaScript value, usually an object or array, to be converted.

\* @param replacer A function that transforms the results.

\* @param space Adds indentation, white space, and line break characters to the return-value JSON text to make it easier to read.

\*/

stringify(value: any, replacer?: (this: any, key: string, value: any) => any, space?: string | number): string;

/\*\*

\* Converts a JavaScript value to a JavaScript Object Notation (JSON) string.

\* @param value A JavaScript value, usually an object or array, to be converted.

\* @param replacer An array of strings and numbers that acts as a approved list for selecting the object properties that will be stringified.

\* @param space Adds indentation, white space, and line break characters to the return-value JSON text to make it easier to read.

\*/

stringify(value: any, replacer?: (number | string)[] | null, space?: string | number): string;

}

/\*\*

\* An intrinsic object that provides functions to convert JavaScript values to and from the JavaScript Object Notation (JSON) format.

\*/

declare var JSON: JSON;

/////////////////////////////

/// ECMAScript Array API (specially handled by compiler)

/////////////////////////////

interface ReadonlyArray<T> {

/\*\*

\* Gets the length of the array. This is a number one higher than the highest element defined in an array.

\*/

readonly length: number;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\*

\* Returns a string representation of an array. The elements are converted to string using their toLocalString methods.

\*/

toLocaleString(): string;

/\*\*

\* Combines two or more arrays.

\* @param items Additional items to add to the end of array1.

\*/

concat(...items: ConcatArray<T>[]): T[];

/\*\*

\* Combines two or more arrays.

\* @param items Additional items to add to the end of array1.

\*/

concat(...items: (T | ConcatArray<T>)[]): T[];

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): T[];

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the search starts at index 0.

\*/

indexOf(searchElement: T, fromIndex?: number): number;

/\*\*

\* Returns the index of the last occurrence of a specified value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the search starts at the last index in the array.

\*/

lastIndexOf(searchElement: T, fromIndex?: number): number;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: T, index: number, array: readonly T[]) => unknown, thisArg?: any): boolean;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: T, index: number, array: readonly T[]) => unknown, thisArg?: any): boolean;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: T, index: number, array: readonly T[]) => void, thisArg?: any): void;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

map<U>(callbackfn: (value: T, index: number, array: readonly T[]) => U, thisArg?: any): U[];

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

filter<S extends T>(callbackfn: (value: T, index: number, array: readonly T[]) => value is S, thisArg?: any): S[];

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: T, index: number, array: readonly T[]) => unknown, thisArg?: any): T[];

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: readonly T[]) => T): T;

reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: readonly T[]) => T, initialValue: T): T;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: T, currentIndex: number, array: readonly T[]) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: readonly T[]) => T): T;

reduceRight(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: readonly T[]) => T, initialValue: T): T;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: T, currentIndex: number, array: readonly T[]) => U, initialValue: U): U;

readonly [n: number]: T;

}

interface ConcatArray<T> {

readonly length: number;

readonly [n: number]: T;

join(separator?: string): string;

slice(start?: number, end?: number): T[];

}

interface Array<T> {

/\*\*

\* Gets or sets the length of the array. This is a number one higher than the highest element defined in an array.

\*/

length: number;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\*

\* Returns a string representation of an array. The elements are converted to string using their toLocalString methods.

\*/

toLocaleString(): string;

/\*\*

\* Removes the last element from an array and returns it.

\*/

pop(): T | undefined;

/\*\*

\* Appends new elements to an array, and returns the new length of the array.

\* @param items New elements of the Array.

\*/

push(...items: T[]): number;

/\*\*

\* Combines two or more arrays.

\* @param items Additional items to add to the end of array1.

\*/

concat(...items: ConcatArray<T>[]): T[];

/\*\*

\* Combines two or more arrays.

\* @param items Additional items to add to the end of array1.

\*/

concat(...items: (T | ConcatArray<T>)[]): T[];

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): T[];

/\*\*

\* Removes the first element from an array and returns it.

\*/

shift(): T | undefined;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): T[];

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: T, b: T) => number): this;

/\*\*

\* Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.

\* @param start The zero-based location in the array from which to start removing elements.

\* @param deleteCount The number of elements to remove.

\*/

splice(start: number, deleteCount?: number): T[];

/\*\*

\* Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.

\* @param start The zero-based location in the array from which to start removing elements.

\* @param deleteCount The number of elements to remove.

\* @param items Elements to insert into the array in place of the deleted elements.

\*/

splice(start: number, deleteCount: number, ...items: T[]): T[];

/\*\*

\* Inserts new elements at the start of an array.

\* @param items Elements to insert at the start of the Array.

\*/

unshift(...items: T[]): number;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the search starts at index 0.

\*/

indexOf(searchElement: T, fromIndex?: number): number;

/\*\*

\* Returns the index of the last occurrence of a specified value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the search starts at the last index in the array.

\*/

lastIndexOf(searchElement: T, fromIndex?: number): number;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: T, index: number, array: T[]) => unknown, thisArg?: any): boolean;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: T, index: number, array: T[]) => unknown, thisArg?: any): boolean;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: T, index: number, array: T[]) => void, thisArg?: any): void;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

map<U>(callbackfn: (value: T, index: number, array: T[]) => U, thisArg?: any): U[];

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

filter<S extends T>(callbackfn: (value: T, index: number, array: T[]) => value is S, thisArg?: any): S[];

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function. If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: T, index: number, array: T[]) => unknown, thisArg?: any): T[];

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T): T;

reduce(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T, initialValue: T): T;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: T, currentIndex: number, array: T[]) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T): T;

reduceRight(callbackfn: (previousValue: T, currentValue: T, currentIndex: number, array: T[]) => T, initialValue: T): T;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order. The return value of the callback function is the accumulated result, and is provided as an argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start the accumulation. The first call to the callbackfn function provides this value as an argument instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: T, currentIndex: number, array: T[]) => U, initialValue: U): U;

[n: number]: T;

}

interface ArrayConstructor {

new(arrayLength?: number): any[];

new <T>(arrayLength: number): T[];

new <T>(...items: T[]): T[];

(arrayLength?: number): any[];

<T>(arrayLength: number): T[];

<T>(...items: T[]): T[];

isArray(arg: any): arg is any[];

readonly prototype: any[];

}

declare var Array: ArrayConstructor;

interface TypedPropertyDescriptor<T> {

enumerable?: boolean;

configurable?: boolean;

writable?: boolean;

value?: T;

get?: () => T;

set?: (value: T) => void;

}

declare type ClassDecorator = <TFunction extends Function>(target: TFunction) => TFunction | void;

declare type PropertyDecorator = (target: Object, propertyKey: string | symbol) => void;

declare type MethodDecorator = <T>(target: Object, propertyKey: string | symbol, descriptor: TypedPropertyDescriptor<T>) => TypedPropertyDescriptor<T> | void;

declare type ParameterDecorator = (target: Object, propertyKey: string | symbol, parameterIndex: number) => void;

declare type PromiseConstructorLike = new <T>(executor: (resolve: (value?: T | PromiseLike<T>) => void, reject: (reason?: any) => void) => void) => PromiseLike<T>;

interface PromiseLike<T> {

/\*\*

\* Attaches callbacks for the resolution and/or rejection of the Promise.

\* @param onfulfilled The callback to execute when the Promise is resolved.

\* @param onrejected The callback to execute when the Promise is rejected.

\* @returns A Promise for the completion of which ever callback is executed.

\*/

then<TResult1 = T, TResult2 = never>(onfulfilled?: ((value: T) => TResult1 | PromiseLike<TResult1>) | undefined | null, onrejected?: ((reason: any) => TResult2 | PromiseLike<TResult2>) | undefined | null): PromiseLike<TResult1 | TResult2>;

}

/\*\*

\* Represents the completion of an asynchronous operation

\*/

interface Promise<T> {

/\*\*

\* Attaches callbacks for the resolution and/or rejection of the Promise.

\* @param onfulfilled The callback to execute when the Promise is resolved.

\* @param onrejected The callback to execute when the Promise is rejected.

\* @returns A Promise for the completion of which ever callback is executed.

\*/

then<TResult1 = T, TResult2 = never>(onfulfilled?: ((value: T) => TResult1 | PromiseLike<TResult1>) | undefined | null, onrejected?: ((reason: any) => TResult2 | PromiseLike<TResult2>) | undefined | null): Promise<TResult1 | TResult2>;

/\*\*

\* Attaches a callback for only the rejection of the Promise.

\* @param onrejected The callback to execute when the Promise is rejected.

\* @returns A Promise for the completion of the callback.

\*/

catch<TResult = never>(onrejected?: ((reason: any) => TResult | PromiseLike<TResult>) | undefined | null): Promise<T | TResult>;

}

interface ArrayLike<T> {

readonly length: number;

readonly [n: number]: T;

}

/\*\*

\* Make all properties in T optional

\*/

type Partial<T> = {

[P in keyof T]?: T[P];

};

/\*\*

\* Make all properties in T required

\*/

type Required<T> = {

[P in keyof T]-?: T[P];

};

/\*\*

\* Make all properties in T readonly

\*/

type Readonly<T> = {

readonly [P in keyof T]: T[P];

};

/\*\*

\* From T, pick a set of properties whose keys are in the union K

\*/

type Pick<T, K extends keyof T> = {

[P in K]: T[P];

};

/\*\*

\* Construct a type with a set of properties K of type T

\*/

type Record<K extends keyof any, T> = {

[P in K]: T;

};

/\*\*

\* Exclude from T those types that are assignable to U

\*/

type Exclude<T, U> = T extends U ? never : T;

/\*\*

\* Extract from T those types that are assignable to U

\*/

type Extract<T, U> = T extends U ? T : never;

/\*\*

\* Construct a type with the properties of T except for those in type K.

\*/

type Omit<T, K extends keyof any> = Pick<T, Exclude<keyof T, K>>;

/\*\*

\* Exclude null and undefined from T

\*/

type NonNullable<T> = T extends null | undefined ? never : T;

/\*\*

\* Obtain the parameters of a function type in a tuple

\*/

type Parameters<T extends (...args: any) => any> = T extends (...args: infer P) => any ? P : never;

/\*\*

\* Obtain the parameters of a constructor function type in a tuple

\*/

type ConstructorParameters<T extends new (...args: any) => any> = T extends new (...args: infer P) => any ? P : never;

/\*\*

\* Obtain the return type of a function type

\*/

type ReturnType<T extends (...args: any) => any> = T extends (...args: any) => infer R ? R : any;

/\*\*

\* Obtain the return type of a constructor function type

\*/

type InstanceType<T extends new (...args: any) => any> = T extends new (...args: any) => infer R ? R : any;

/\*\*

\* Marker for contextual 'this' type

\*/

interface ThisType<T> { }

/\*\*

\* Represents a raw buffer of binary data, which is used to store data for the

\* different typed arrays. ArrayBuffers cannot be read from or written to directly,

\* but can be passed to a typed array or DataView Object to interpret the raw

\* buffer as needed.

\*/

interface ArrayBuffer {

/\*\*

\* Read-only. The length of the ArrayBuffer (in bytes).

\*/

readonly byteLength: number;

/\*\*

\* Returns a section of an ArrayBuffer.

\*/

slice(begin: number, end?: number): ArrayBuffer;

}

/\*\*

\* Allowed ArrayBuffer types for the buffer of an ArrayBufferView and related Typed Arrays.

\*/

interface ArrayBufferTypes {

ArrayBuffer: ArrayBuffer;

}

type ArrayBufferLike = ArrayBufferTypes[keyof ArrayBufferTypes];

interface ArrayBufferConstructor {

readonly prototype: ArrayBuffer;

new(byteLength: number): ArrayBuffer;

isView(arg: any): arg is ArrayBufferView;

}

declare var ArrayBuffer: ArrayBufferConstructor;

interface ArrayBufferView {

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

byteOffset: number;

}

interface DataView {

readonly buffer: ArrayBuffer;

readonly byteLength: number;

readonly byteOffset: number;

/\*\*

\* Gets the Float32 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getFloat32(byteOffset: number, littleEndian?: boolean): number;

/\*\*

\* Gets the Float64 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getFloat64(byteOffset: number, littleEndian?: boolean): number;

/\*\*

\* Gets the Int8 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getInt8(byteOffset: number): number;

/\*\*

\* Gets the Int16 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getInt16(byteOffset: number, littleEndian?: boolean): number;

/\*\*

\* Gets the Int32 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getInt32(byteOffset: number, littleEndian?: boolean): number;

/\*\*

\* Gets the Uint8 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getUint8(byteOffset: number): number;

/\*\*

\* Gets the Uint16 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getUint16(byteOffset: number, littleEndian?: boolean): number;

/\*\*

\* Gets the Uint32 value at the specified byte offset from the start of the view. There is

\* no alignment constraint; multi-byte values may be fetched from any offset.

\* @param byteOffset The place in the buffer at which the value should be retrieved.

\*/

getUint32(byteOffset: number, littleEndian?: boolean): number;

/\*\*

\* Stores an Float32 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\* @param littleEndian If false or undefined, a big-endian value should be written,

\* otherwise a little-endian value should be written.

\*/

setFloat32(byteOffset: number, value: number, littleEndian?: boolean): void;

/\*\*

\* Stores an Float64 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\* @param littleEndian If false or undefined, a big-endian value should be written,

\* otherwise a little-endian value should be written.

\*/

setFloat64(byteOffset: number, value: number, littleEndian?: boolean): void;

/\*\*

\* Stores an Int8 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\*/

setInt8(byteOffset: number, value: number): void;

/\*\*

\* Stores an Int16 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\* @param littleEndian If false or undefined, a big-endian value should be written,

\* otherwise a little-endian value should be written.

\*/

setInt16(byteOffset: number, value: number, littleEndian?: boolean): void;

/\*\*

\* Stores an Int32 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\* @param littleEndian If false or undefined, a big-endian value should be written,

\* otherwise a little-endian value should be written.

\*/

setInt32(byteOffset: number, value: number, littleEndian?: boolean): void;

/\*\*

\* Stores an Uint8 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\*/

setUint8(byteOffset: number, value: number): void;

/\*\*

\* Stores an Uint16 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\* @param littleEndian If false or undefined, a big-endian value should be written,

\* otherwise a little-endian value should be written.

\*/

setUint16(byteOffset: number, value: number, littleEndian?: boolean): void;

/\*\*

\* Stores an Uint32 value at the specified byte offset from the start of the view.

\* @param byteOffset The place in the buffer at which the value should be set.

\* @param value The value to set.

\* @param littleEndian If false or undefined, a big-endian value should be written,

\* otherwise a little-endian value should be written.

\*/

setUint32(byteOffset: number, value: number, littleEndian?: boolean): void;

}

interface DataViewConstructor {

new(buffer: ArrayBufferLike, byteOffset?: number, byteLength?: number): DataView;

}

declare var DataView: DataViewConstructor;

/\*\*

\* A typed array of 8-bit integer values. The contents are initialized to 0. If the requested

\* number of bytes could not be allocated an exception is raised.

\*/

interface Int8Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Int8Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Int8Array) => any, thisArg?: any): Int8Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Int8Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Int8Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Int8Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Int8Array) => number, thisArg?: any): Int8Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int8Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int8Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Int8Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int8Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int8Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Int8Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Int8Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Int8Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Int8Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Int8Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Int8Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Int8Array;

[index: number]: number;

}

interface Int8ArrayConstructor {

readonly prototype: Int8Array;

new(length: number): Int8Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Int8Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Int8Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Int8Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Int8Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Int8Array;

}

declare var Int8Array: Int8ArrayConstructor;

/\*\*

\* A typed array of 8-bit unsigned integer values. The contents are initialized to 0. If the

\* requested number of bytes could not be allocated an exception is raised.

\*/

interface Uint8Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Uint8Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Uint8Array) => any, thisArg?: any): Uint8Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Uint8Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Uint8Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Uint8Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Uint8Array) => number, thisArg?: any): Uint8Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint8Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint8Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Uint8Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Uint8Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Uint8Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Uint8Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Uint8Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Uint8Array;

[index: number]: number;

}

interface Uint8ArrayConstructor {

readonly prototype: Uint8Array;

new(length: number): Uint8Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Uint8Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Uint8Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Uint8Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Uint8Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Uint8Array;

}

declare var Uint8Array: Uint8ArrayConstructor;

/\*\*

\* A typed array of 8-bit unsigned integer (clamped) values. The contents are initialized to 0.

\* If the requested number of bytes could not be allocated an exception is raised.

\*/

interface Uint8ClampedArray {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Uint8ClampedArray) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Uint8ClampedArray) => any, thisArg?: any): Uint8ClampedArray;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Uint8ClampedArray) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Uint8ClampedArray) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Uint8ClampedArray) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Uint8ClampedArray) => number, thisArg?: any): Uint8ClampedArray;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8ClampedArray) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8ClampedArray) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint8ClampedArray) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8ClampedArray) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint8ClampedArray) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint8ClampedArray) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Uint8ClampedArray;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Uint8ClampedArray;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Uint8ClampedArray) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Uint8ClampedArray view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Uint8ClampedArray;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Uint8ClampedArray;

[index: number]: number;

}

interface Uint8ClampedArrayConstructor {

readonly prototype: Uint8ClampedArray;

new(length: number): Uint8ClampedArray;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Uint8ClampedArray;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Uint8ClampedArray;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Uint8ClampedArray;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Uint8ClampedArray;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Uint8ClampedArray;

}

declare var Uint8ClampedArray: Uint8ClampedArrayConstructor;

/\*\*

\* A typed array of 16-bit signed integer values. The contents are initialized to 0. If the

\* requested number of bytes could not be allocated an exception is raised.

\*/

interface Int16Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Int16Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Int16Array) => any, thisArg?: any): Int16Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Int16Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Int16Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Int16Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Int16Array) => number, thisArg?: any): Int16Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int16Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int16Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Int16Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int16Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int16Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Int16Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Int16Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Int16Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Int16Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Int16Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Int16Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Int16Array;

[index: number]: number;

}

interface Int16ArrayConstructor {

readonly prototype: Int16Array;

new(length: number): Int16Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Int16Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Int16Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Int16Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Int16Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Int16Array;

}

declare var Int16Array: Int16ArrayConstructor;

/\*\*

\* A typed array of 16-bit unsigned integer values. The contents are initialized to 0. If the

\* requested number of bytes could not be allocated an exception is raised.

\*/

interface Uint16Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Uint16Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Uint16Array) => any, thisArg?: any): Uint16Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Uint16Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Uint16Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Uint16Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Uint16Array) => number, thisArg?: any): Uint16Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint16Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint16Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint16Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint16Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint16Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint16Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Uint16Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Uint16Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Uint16Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Uint16Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Uint16Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Uint16Array;

[index: number]: number;

}

interface Uint16ArrayConstructor {

readonly prototype: Uint16Array;

new(length: number): Uint16Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Uint16Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Uint16Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Uint16Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Uint16Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Uint16Array;

}

declare var Uint16Array: Uint16ArrayConstructor;

/\*\*

\* A typed array of 32-bit signed integer values. The contents are initialized to 0. If the

\* requested number of bytes could not be allocated an exception is raised.

\*/

interface Int32Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Int32Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Int32Array) => any, thisArg?: any): Int32Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Int32Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Int32Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Int32Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Int32Array) => number, thisArg?: any): Int32Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int32Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int32Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Int32Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int32Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Int32Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Int32Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Int32Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Int32Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Int32Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Int32Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Int32Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Int32Array;

[index: number]: number;

}

interface Int32ArrayConstructor {

readonly prototype: Int32Array;

new(length: number): Int32Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Int32Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Int32Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Int32Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Int32Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Int32Array;

}

declare var Int32Array: Int32ArrayConstructor;

/\*\*

\* A typed array of 32-bit unsigned integer values. The contents are initialized to 0. If the

\* requested number of bytes could not be allocated an exception is raised.

\*/

interface Uint32Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Uint32Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Uint32Array) => any, thisArg?: any): Uint32Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Uint32Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Uint32Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Uint32Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Uint32Array) => number, thisArg?: any): Uint32Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint32Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint32Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint32Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint32Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Uint32Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Uint32Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Uint32Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Uint32Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Uint32Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Uint32Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Uint32Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Uint32Array;

[index: number]: number;

}

interface Uint32ArrayConstructor {

readonly prototype: Uint32Array;

new(length: number): Uint32Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Uint32Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Uint32Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Uint32Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Uint32Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Uint32Array;

}

declare var Uint32Array: Uint32ArrayConstructor;

/\*\*

\* A typed array of 32-bit float values. The contents are initialized to 0. If the requested number

\* of bytes could not be allocated an exception is raised.

\*/

interface Float32Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Float32Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Float32Array) => any, thisArg?: any): Float32Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Float32Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Float32Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Float32Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Float32Array) => number, thisArg?: any): Float32Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float32Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float32Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Float32Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float32Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float32Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Float32Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Float32Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Float32Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Float32Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* Gets a new Float32Array view of the ArrayBuffer store for this array, referencing the elements

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Float32Array;

/\*\*

\* Converts a number to a string by using the current locale.

\*/

toLocaleString(): string;

/\*\*

\* Returns a string representation of an array.

\*/

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Float32Array;

[index: number]: number;

}

interface Float32ArrayConstructor {

readonly prototype: Float32Array;

new(length: number): Float32Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Float32Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Float32Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Float32Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Float32Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Float32Array;

}

declare var Float32Array: Float32ArrayConstructor;

/\*\*

\* A typed array of 64-bit float values. The contents are initialized to 0. If the requested

\* number of bytes could not be allocated an exception is raised.

\*/

interface Float64Array {

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* The ArrayBuffer instance referenced by the array.

\*/

readonly buffer: ArrayBufferLike;

/\*\*

\* The length in bytes of the array.

\*/

readonly byteLength: number;

/\*\*

\* The offset in bytes of the array.

\*/

readonly byteOffset: number;

/\*\*

\* Returns the this object after copying a section of the array identified by start and end

\* to the same array starting at position target

\* @param target If target is negative, it is treated as length+target where length is the

\* length of the array.

\* @param start If start is negative, it is treated as length+start. If end is negative, it

\* is treated as length+end.

\* @param end If not specified, length of the this object is used as its default value.

\*/

copyWithin(target: number, start: number, end?: number): this;

/\*\*

\* Determines whether all the members of an array satisfy the specified test.

\* @param callbackfn A function that accepts up to three arguments. The every method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value false, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

every(callbackfn: (value: number, index: number, array: Float64Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Returns the this object after filling the section identified by start and end with value

\* @param value value to fill array section with

\* @param start index to start filling the array at. If start is negative, it is treated as

\* length+start where length is the length of the array.

\* @param end index to stop filling the array at. If end is negative, it is treated as

\* length+end.

\*/

fill(value: number, start?: number, end?: number): this;

/\*\*

\* Returns the elements of an array that meet the condition specified in a callback function.

\* @param callbackfn A function that accepts up to three arguments. The filter method calls

\* the callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

filter(callbackfn: (value: number, index: number, array: Float64Array) => any, thisArg?: any): Float64Array;

/\*\*

\* Returns the value of the first element in the array where predicate is true, and undefined

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found, find

\* immediately returns that element value. Otherwise, find returns undefined.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

find(predicate: (value: number, index: number, obj: Float64Array) => boolean, thisArg?: any): number | undefined;

/\*\*

\* Returns the index of the first element in the array where predicate is true, and -1

\* otherwise.

\* @param predicate find calls predicate once for each element of the array, in ascending

\* order, until it finds one where predicate returns true. If such an element is found,

\* findIndex immediately returns that element index. Otherwise, findIndex returns -1.

\* @param thisArg If provided, it will be used as the this value for each invocation of

\* predicate. If it is not provided, undefined is used instead.

\*/

findIndex(predicate: (value: number, index: number, obj: Float64Array) => boolean, thisArg?: any): number;

/\*\*

\* Performs the specified action for each element in an array.

\* @param callbackfn A function that accepts up to three arguments. forEach calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

forEach(callbackfn: (value: number, index: number, array: Float64Array) => void, thisArg?: any): void;

/\*\*

\* Returns the index of the first occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

indexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* Adds all the elements of an array separated by the specified separator string.

\* @param separator A string used to separate one element of an array from the next in the

\* resulting String. If omitted, the array elements are separated with a comma.

\*/

join(separator?: string): string;

/\*\*

\* Returns the index of the last occurrence of a value in an array.

\* @param searchElement The value to locate in the array.

\* @param fromIndex The array index at which to begin the search. If fromIndex is omitted, the

\* search starts at index 0.

\*/

lastIndexOf(searchElement: number, fromIndex?: number): number;

/\*\*

\* The length of the array.

\*/

readonly length: number;

/\*\*

\* Calls a defined callback function on each element of an array, and returns an array that

\* contains the results.

\* @param callbackfn A function that accepts up to three arguments. The map method calls the

\* callbackfn function one time for each element in the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

map(callbackfn: (value: number, index: number, array: Float64Array) => number, thisArg?: any): Float64Array;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float64Array) => number): number;

reduce(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float64Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array. The return value of

\* the callback function is the accumulated result, and is provided as an argument in the next

\* call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduce method calls the

\* callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduce<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Float64Array) => U, initialValue: U): U;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an

\* argument instead of an array value.

\*/

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float64Array) => number): number;

reduceRight(callbackfn: (previousValue: number, currentValue: number, currentIndex: number, array: Float64Array) => number, initialValue: number): number;

/\*\*

\* Calls the specified callback function for all the elements in an array, in descending order.

\* The return value of the callback function is the accumulated result, and is provided as an

\* argument in the next call to the callback function.

\* @param callbackfn A function that accepts up to four arguments. The reduceRight method calls

\* the callbackfn function one time for each element in the array.

\* @param initialValue If initialValue is specified, it is used as the initial value to start

\* the accumulation. The first call to the callbackfn function provides this value as an argument

\* instead of an array value.

\*/

reduceRight<U>(callbackfn: (previousValue: U, currentValue: number, currentIndex: number, array: Float64Array) => U, initialValue: U): U;

/\*\*

\* Reverses the elements in an Array.

\*/

reverse(): Float64Array;

/\*\*

\* Sets a value or an array of values.

\* @param array A typed or untyped array of values to set.

\* @param offset The index in the current array at which the values are to be written.

\*/

set(array: ArrayLike<number>, offset?: number): void;

/\*\*

\* Returns a section of an array.

\* @param start The beginning of the specified portion of the array.

\* @param end The end of the specified portion of the array. This is exclusive of the element at the index 'end'.

\*/

slice(start?: number, end?: number): Float64Array;

/\*\*

\* Determines whether the specified callback function returns true for any element of an array.

\* @param callbackfn A function that accepts up to three arguments. The some method calls

\* the callbackfn function for each element in the array until the callbackfn returns a value

\* which is coercible to the Boolean value true, or until the end of the array.

\* @param thisArg An object to which the this keyword can refer in the callbackfn function.

\* If thisArg is omitted, undefined is used as the this value.

\*/

some(callbackfn: (value: number, index: number, array: Float64Array) => unknown, thisArg?: any): boolean;

/\*\*

\* Sorts an array.

\* @param compareFn Function used to determine the order of the elements. It is expected to return

\* a negative value if first argument is less than second argument, zero if they're equal and a positive

\* value otherwise. If omitted, the elements are sorted in ascending, ASCII character order.

\* ```ts

\* [11,2,22,1].sort((a, b) => a - b)

\* ```

\*/

sort(compareFn?: (a: number, b: number) => number): this;

/\*\*

\* at begin, inclusive, up to end, exclusive.

\* @param begin The index of the beginning of the array.

\* @param end The index of the end of the array.

\*/

subarray(begin?: number, end?: number): Float64Array;

toString(): string;

/\*\* Returns the primitive value of the specified object. \*/

valueOf(): Float64Array;

[index: number]: number;

}

interface Float64ArrayConstructor {

readonly prototype: Float64Array;

new(length: number): Float64Array;

new(arrayOrArrayBuffer: ArrayLike<number> | ArrayBufferLike): Float64Array;

new(buffer: ArrayBufferLike, byteOffset: number, length?: number): Float64Array;

/\*\*

\* The size in bytes of each element in the array.

\*/

readonly BYTES\_PER\_ELEMENT: number;

/\*\*

\* Returns a new array from a set of elements.

\* @param items A set of elements to include in the new array object.

\*/

of(...items: number[]): Float64Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\*/

from(arrayLike: ArrayLike<number>): Float64Array;

/\*\*

\* Creates an array from an array-like or iterable object.

\* @param arrayLike An array-like or iterable object to convert to an array.

\* @param mapfn A mapping function to call on every element of the array.

\* @param thisArg Value of 'this' used to invoke the mapfn.

\*/

from<T>(arrayLike: ArrayLike<T>, mapfn: (v: T, k: number) => number, thisArg?: any): Float64Array;

}

declare var Float64Array: Float64ArrayConstructor;

/////////////////////////////

/// ECMAScript Internationalization API

/////////////////////////////

declare namespace Intl {

interface CollatorOptions {

usage?: string;

localeMatcher?: string;

numeric?: boolean;

caseFirst?: string;

sensitivity?: string;

ignorePunctuation?: boolean;

}

interface ResolvedCollatorOptions {

locale: string;

usage: string;

sensitivity: string;

ignorePunctuation: boolean;

collation: string;

caseFirst: string;

numeric: boolean;

}

interface Collator {

compare(x: string, y: string): number;

resolvedOptions(): ResolvedCollatorOptions;

}

var Collator: {

new(locales?: string | string[], options?: CollatorOptions): Collator;

(locales?: string | string[], options?: CollatorOptions): Collator;

supportedLocalesOf(locales: string | string[], options?: CollatorOptions): string[];

};

interface NumberFormatOptions {

localeMatcher?: string;

style?: string;

currency?: string;

currencyDisplay?: string;

useGrouping?: boolean;

minimumIntegerDigits?: number;

minimumFractionDigits?: number;

maximumFractionDigits?: number;

minimumSignificantDigits?: number;

maximumSignificantDigits?: number;

}

interface ResolvedNumberFormatOptions {

locale: string;

numberingSystem: string;

style: string;

currency?: string;

currencyDisplay?: string;

minimumIntegerDigits: number;

minimumFractionDigits: number;

maximumFractionDigits: number;

minimumSignificantDigits?: number;

maximumSignificantDigits?: number;

useGrouping: boolean;

}

interface NumberFormat {

format(value: number): string;

resolvedOptions(): ResolvedNumberFormatOptions;

}

var NumberFormat: {

new(locales?: string | string[], options?: NumberFormatOptions): NumberFormat;

(locales?: string | string[], options?: NumberFormatOptions): NumberFormat;

supportedLocalesOf(locales: string | string[], options?: NumberFormatOptions): string[];

};

interface DateTimeFormatOptions {

localeMatcher?: string;

weekday?: string;

era?: string;

year?: string;

month?: string;

day?: string;

hour?: string;

minute?: string;

second?: string;

timeZoneName?: string;

formatMatcher?: string;

hour12?: boolean;

timeZone?: string;

}

interface ResolvedDateTimeFormatOptions {

locale: string;

calendar: string;

numberingSystem: string;

timeZone: string;

hour12?: boolean;

weekday?: string;

era?: string;

year?: string;

month?: string;

day?: string;

hour?: string;

minute?: string;

second?: string;

timeZoneName?: string;

}

interface DateTimeFormat {

format(date?: Date | number): string;

resolvedOptions(): ResolvedDateTimeFormatOptions;

}

var DateTimeFormat: {

new(locales?: string | string[], options?: DateTimeFormatOptions): DateTimeFormat;

(locales?: string | string[], options?: DateTimeFormatOptions): DateTimeFormat;

supportedLocalesOf(locales: string | string[], options?: DateTimeFormatOptions): string[];

};

}

interface String {

/\*\*

\* Determines whether two strings are equivalent in the current or specified locale.

\* @param that String to compare to target string

\* @param locales A locale string or array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used. This parameter must conform to BCP 47 standards; see the Intl.Collator object for details.

\* @param options An object that contains one or more properties that specify comparison options. see the Intl.Collator object for details.

\*/

localeCompare(that: string, locales?: string | string[], options?: Intl.CollatorOptions): number;

}

interface Number {

/\*\*

\* Converts a number to a string by using the current or specified locale.

\* @param locales A locale string or array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

\* @param options An object that contains one or more properties that specify comparison options.

\*/

toLocaleString(locales?: string | string[], options?: Intl.NumberFormatOptions): string;

}

interface Date {

/\*\*

\* Converts a date and time to a string by using the current or specified locale.

\* @param locales A locale string or array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

\* @param options An object that contains one or more properties that specify comparison options.

\*/

toLocaleString(locales?: string | string[], options?: Intl.DateTimeFormatOptions): string;

/\*\*

\* Converts a date to a string by using the current or specified locale.

\* @param locales A locale string or array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

\* @param options An object that contains one or more properties that specify comparison options.

\*/

toLocaleDateString(locales?: string | string[], options?: Intl.DateTimeFormatOptions): string;

/\*\*

\* Converts a time to a string by using the current or specified locale.

\* @param locales A locale string or array of locale strings that contain one or more language or locale tags. If you include more than one locale string, list them in descending order of priority so that the first entry is the preferred locale. If you omit this parameter, the default locale of the JavaScript runtime is used.

\* @param options An object that contains one or more properties that specify comparison options.

\*/

toLocaleTimeString(locales?: string | string[], options?: Intl.DateTimeFormatOptions): string;

}