/////////////////////////////  
/// ECMAScript Array API (specially handled by compiler)  
/////////////////////////////  
  
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;  
 toLocaleString(): string;

----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------  
 /\*\*  
 \* 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;

----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------  
 /\*\*  
 \* Removes the last element from an array and returns it. This method changes the length of the array.  
 \*/  
 pop(): T;

----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------  
 /\*\*  
 \* Combines two or more arrays.  
 \* **@param** items Additional items to add to the end of array1.  
 \*/  
 concat<U extends T[]>(...*items*: U[]): T[];

----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------  
 /\*\*  
 \* Combines two or more arrays.  
 \* **@param** items Additional items to add to the end of array1.  
 \*/  
 concat(...*items*: 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;

----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------  
 /\*\*   
 \* 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.  
 \*/  
 slice(*start*?: number, *end*?: number): T[];  
----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------  
 /\*\*  
 \* Sorts an array.  
 \* **@param** compareFn The name of the function used to determine the order of the elements. If omitted, the elements are sorted in ascending, ASCII character order.  
 \*/  
 sort(*compareFn*?: (*a*: T, *b*: T) => 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.  
 \*/  
 splice(*start*: 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 array1 until the callbackfn returns 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[]) => boolean, *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 array1 until the callbackfn returns 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[]) => boolean, *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(*callbackfn*: (*value*: T, *index*: number, *array*: T[]) => boolean, *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, *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, *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;  
}

}

----------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------