#### Array.prototype.indexOf ( searchElement [ , fromIndex ] )

**indexOf** compares *searchElement* to the elements of the array, in ascending order, using the Strict Equality Comparison algorithm (11.9.1), and if found at one or more positions, returns the index of the first such position; otherwise, -1 is returned.

The optional second argument *fromIndex* defaults to 0 (i.e. the whole array is searched). If it is greater than or equal to the length of the array, -1 is returned, i.e. the array will not be searched. If it is negative, it is used as the offset from the end of the array to compute *fromIndex*. If the computed index is less than 0, the whole array will be searched.

When the **indexOf** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, **"length"**).
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If *len* is 0, return -1.
7. If argument *fromIndex* was passed let *n* be ToInteger(*fromIndex*); else let *n* be 0.
8. ReturnIfAbrupt(*n*).
9. If *n* ≥ *len*, return -1.
10. If *n* ≥ 0, then
    1. Let *k* be *n*.
11. Else *n*<0,
    1. Let *k* be *len* - abs(*n*).
    2. If *k* < 0, then let *k* be 0.
12. Repeat, while *k*<*len*
    1. Let *kPresent* be the result of HasProperty(*O*, ToString(*k*)).
    2. ReturnIfAbrupt(*kPresent*).
    3. If *kPresent* is **true**, then
       1. Let *elementK* be the result of Get(*O*, ToString(*k*)).
       2. ReturnIfAbrupt(*elementK*).
       3. Let *same* be the result of performing Strict Equality Comparison *searchElement* === *elementK*.
       4. If *same* is **true**, return *k*.
    4. Increase *k* by 1.
13. Return -1.

The **length** property of the **indexOf** method is **1**.

NOTE The **indexOf** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **indexOf** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.

#### Array.prototype.lastIndexOf ( searchElement [ , fromIndex ] )

**lastIndexOf** compares *searchElement* to the elements of the array in descending order using the Strict Equality Comparison algorithm (11.9.1), and if found at one or more positions, returns the index of the last such position; otherwise, -1 is returned.

The optional second argument *fromIndex* defaults to the array's length minus one (i.e. the whole array is searched). If it is greater than or equal to the length of the array, the whole array will be searched. If it is negative, it is used as the offset from the end of the array to compute *fromIndex*. If the computed index is less than 0, -1 is returned.

When the **lastIndexOf** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, "length")
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If *len* is 0, return -1.
7. If argument *fromIndex* was passed let *n* be ToInteger(*fromIndex*); else let *n* be *len*-1.
8. ReturnIfAbrupt(*n*).
9. If *n* ≥ 0, then let *k* be min(*n*, *len* – 1).
10. Else *n* < 0,
    1. Let *k* be *len* - abs(*n*).
11. Repeat, while *k* ≥ 0
    1. Let *kPresent* be the result of HasProperty(*O*, ToString(*k*)).
    2. ReturnIfAbrupt(*kPresent*).
    3. If *kPresent* is true, then
       1. Let *elementK* be the result of Get(*O*, ToString(*k*)).
       2. ReturnIfAbrupt(*elementK*).
       3. Let *same* be the result of performing Strict Equality Comparison *searchElement* === *elementK*.
       4. If *same* is true, return *k*.
    4. Decrease *k* by 1.
12. Return -1.

The **length** property of the **lastIndexOf** method is **1**.

NOTE The **lastIndexOf** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **lastIndexOf** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.

#### Array.prototype.every ( callbackfn [ , thisArg ] )

*callbackfn* should be a function that accepts three arguments and returns a value that is coercible to the Boolean value **true** or **false**. **every** calls *callbackfn* once for each element present in the array, in ascending order, until it finds one where *callbackfn* returns **false**. If such an element is found, **every** immediately returns **false**. Otherwise, if *callbackfn* returned **true** for all elements, **every** will return **true**. *callbackfn* is called only for elements of the array which actually exist; it is not called for missing elements of the array.

If a *thisArg* parameter is provided, it will be used as the **this** value for each invocation of *callbackfn*. If it is not provided, **undefined** is used instead.

*callbackfn* is called with three arguments: the value of the element, the index of the element, and the object being traversed.

**every** does not directly mutate the object on which it is called but the object may be mutated by the calls to *callbackfn*.

The range of elements processed by **every** is set before the first call to *callbackfn*. Elements which are appended to the array after the call to **every** begins will not be visited by *callbackfn*. If existing elements of the array are changed, their value as passed to *callbackfn* will be the value at the time **every** visits them; elements that are deleted after the call to **every** begins and before being visited are not visited. **every** acts like the "for all" quantifier in mathematics. In particular, for an empty array, it returns true.

When the **every** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, "length")
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If IsCallable(*callbackfn*) is false, throw a **TypeError** exception.
7. If *thisArg* was supplied, let *T* be *thisArg*; else let *T* be **undefined**.
8. Let *k* be 0.
9. Repeat, while *k* < *len*
   1. Let *Pk* be ToString(*k*).
   2. Let *kPresent* be the result of HasProperty(*O*, *Pk*).
   3. ReturnIfAbrupt(*kPresent*).
   4. If *kPresent* is true, then
      1. Let *kValue* be the result of Get(*O*, *Pk*).
      2. ReturnIfAbrupt(*kValue*).
      3. Let *testResult* be the result of calling the [[Call]] internal method of *callbackfn* with *T* as *thisArgument* and a List containing *kValue*, *k*, and *O* as *argumentsList*.
      4. ReturnIfAbrupt(*testResult*).
      5. If ToBoolean(*testResult*) is false, return false.
   5. Increase *k* by 1.
10. Return true.

The **length** property of the **every** method is **1**.

NOTE The **every** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **every** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.

#### Array.prototype.some ( callbackfn [ , thisArg ] )

*callbackfn* should be a function that accepts three arguments and returns a value that is coercible to the Boolean value **true** or **false**. **some** calls *callbackfn* once for each element present in the array, in ascending order, until it finds one where *callbackfn* returns **true**. If such an element is found, **some** immediately returns **true**. Otherwise, **some** returns **false**. *callbackfn* is called only for elements of the array which actually exist; it is not called for missing elements of the array.

If a *thisArg* parameter is provided, it will be used as the **this** value for each invocation of *callbackfn*. If it is not provided, **undefined** is used instead.

*callbackfn* is called with three arguments: the value of the element, the index of the element, and the object being traversed.

**some** does not directly mutate the object on which it is called but the object may be mutated by the calls to *callbackfn*.

The range of elements processed by **some** is set before the first call to *callbackfn*. Elements that are appended to the array after the call to **some** begins will not be visited by *callbackfn*. If existing elements of the array are changed, their value as passed to *callbackfn* will be the value at the time that **some** visits them; elements that are deleted after the call to **some** begins and before being visited are not visited. **some** acts like the "exists" quantifier in mathematics. In particular, for an empty array, it returns **false**.

When the **some** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, **"length"**).
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If IsCallable(*callbackfn*) is **false**, throw a **TypeError** exception.
7. If *thisArg* was supplied, let *T* be *thisArg*; else let *T* be **undefined**.
8. Let *k* be 0.
9. Repeat, while *k* < *len*
   1. Let *Pk* be ToString(*k*).
   2. Let *kPresent* be the result of HasProperty(*O*, *Pk*).
   3. ReturnIfAbrupt(*kPresent*).
   4. If *kPresent* is **true**, then
      1. Let *kValue* be the result of Get(*O*, *Pk*).
      2. ReturnIfAbrupt(*kValue*).
      3. Let *testResult* be the result of calling the [[Call]] internal method of *callbackfn* with *T* as *thisArgument* and a List containing *kValue*, *k*, and *O* as *argumentsList*.
      4. ReturnIfAbrupt(*testResult*).
      5. If ToBoolean(*testResult*) is **true**, return **true**.
   5. Increase *k* by 1.
10. Return **false**.

The **length** property of the **some** method is **1**.

NOTE The **some** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **some** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.

#### Array.prototype.forEach ( callbackfn [ , thisArg ] )

*callbackfn* should be a function that accepts three arguments. **forEach** calls *callbackfn* once for each element present in the array, in ascending order. *callbackfn* is called only for elements of the array which actually exist; it is not called for missing elements of the array.

If a *thisArg* parameter is provided, it will be used as the **this** value for each invocation of *callbackfn*. If it is not provided, **undefined** is used instead.

*callbackfn* is called with three arguments: the value of the element, the index of the element, and the object being traversed.

**forEach** does not directly mutate the object on which it is called but the object may be mutated by the calls to *callbackfn*.

The range of elements processed by **forEach** is set before the first call to *callbackfn*. Elements which are appended to the array after the call to **forEach** begins will not be visited by *callbackfn*. If existing elements of the array are changed, their value as passed to callback will be the value at the time **forEach** visits them; elements that are deleted after the call to **forEach** begins and before being visited are not visited.

When the **forEach** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, **"length"**).
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If IsCallable(*callbackfn*) is **false**, throw a **TypeError** exception.
7. If *thisArg* was supplied, let *T* be *thisArg*; else let *T* be **undefined**.
8. Let *k* be 0.
9. Repeat, while *k* < *len*
   1. Let *Pk* be ToString(*k*).
   2. Let *kPresent* be the result of HasProperty(*O*, *Pk*).
   3. ReturnIfAbrupt(*kPresent*).
   4. If *kPresent* is **true**, then
      1. Let *kValue* be the result of Get(*O*, *Pk*).
      2. ReturnIfAbrupt(*kValue*).
      3. Let *funcResult* be the result of calling the [[Call]] internal method of *callbackfn* with *T* as *thisArgument* and a List containing *kValue*, *k*, and *O* as *argumentsList*.
      4. ReturnIfAbrupt(*funcResult*).
   5. Increase *k* by 1.
10. Return **undefined**.

The length property of the **forEach** method is **1**.

NOTE The **forEach** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **forEach** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.

#### Array.prototype.find ( predicate , thisArg = undefined )

*predicate* should be a function that accepts three arguments and returns a value that is coercible to the Boolean value **true** or **false**. **find** calls *predicate* once for each element present in 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**. *predicate* is called only for elements of the array which actually exist; it is not called for missing elements of the array.

If a *thisArg* parameter is provided, it will be used as the **this** value for each invocation of *predicate*. If it is not provided, **undefined** is used instead.

*predicate* is called with three arguments: the value of the element, the index of the element, and the object being traversed.

**find** does not directly mutate the object on which it is called but the object may be mutated by the calls to *predicate*.

The range of elements processed by **find** is set before the first call to callbackfn. Elements that are appended to the array after the call to **find** begins will not be visited by callbackfn. If existing elements of the array are changed, their value as passed to *predicate* will be the value at the time that **find** visits them; elements that are deleted after the call to **find** begins and before being visited are not visited.

When the **find** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, "length").
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If IsCallable(*predicate*) is **false**, throw a **TypeError** exception.
7. If *thisArg* was supplied, let *T* be *thisArg*; else let *T* be **undefined**.
8. Let *k* be 0.
9. Repeat, while *k* < *len*
   1. Let *Pk* be ToString(*k*).
   2. Let *kPresent* be the result of HasProperty(*O*, *Pk*).
   3. ReturnIfAbrupt(*kPresent*).
   4. If *kPresent* is **true**, then
      1. Let *kValue* be the result of Get(*O*, *Pk*).
      2. ReturnIfAbrupt(*kValue*).
      3. Let *testResult* be the result of calling the [[Call]] internal method of *predicate* with *T* as *thisArgument* and a List containing *kValue*, *k*, and *O* as *argumentsList*.
      4. ReturnIfAbrupt(*testResult*).
      5. If ToBoolean(*testResult*) is **true**, return *kValue*.
   5. Increase *k* by 1.
10. Return **undefined**.

The length property of the **find** method is **1**.

NOTE The **find** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **find** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.

#### Array.prototype.findIndex ( predicate , thisArg = undefined )

*predicate* should be a function that accepts three arguments and returns a value that is coercible to the Boolean value **true** or **false**. **findIndex** calls *predicate* once for each element present in the array, in ascending order, until it finds one where *predicate* returns **true**. If such an element is found, **findIndex** immediately returns the index of that element value. Otherwise, **findIndex** returns -1. *predicate* is called only for elements of the array which actually exist; it is not called for missing elements of the array.

If a *thisArg* parameter is provided, it will be used as the **this** value for each invocation of *predicate*. If it is not provided, *undefined* is used instead.

*predicate* is called with three arguments: the value of the element, the index of the element, and the object being traversed.

**findIndex** does not directly mutate the object on which it is called but the object may be mutated by the calls to *predicate*.

The range of elements processed by **findIndex** is set before the first call to callbackfn. Elements that are appended to the array after the call to **findIndex** begins will not be visited by callbackfn. If existing elements of the array are changed, their value as passed to *predicate* will be the value at the time that **findIndex** visits them; elements that are deleted after the call to **findIndex** begins and before being visited are not visited.

When the **findIndex** method is called with one or two arguments, the following steps are taken:

1. Let *O* be the result of calling ToObject passing the **this** value as the argument.
2. ReturnIfAbrupt(*O*).
3. Let *lenValue* be the result of Get(*O*, "length").
4. Let *len* be ToLength(*lenValue*).
5. ReturnIfAbrupt(*len*).
6. If IsCallable(*predicate*) is **false**, throw a **TypeError** exception.
7. If *thisArg* was supplied, let *T* be *thisArg*; else let *T* be **undefined**.
8. Let *k* be 0.
9. Repeat, while *k* < *len*
   1. Let *Pk* be ToString(*k*).
   2. Let *kPresent* be the result of HasProperty(*O*, *Pk*).
   3. ReturnIfAbrupt(*kPresent*).
   4. If *kPresent* is **true**, then
      1. Let *kValue* be the result of Get(*O*, *Pk*).
      2. ReturnIfAbrupt(*kValue*).
      3. Let *testResult* be the result of calling the [[Call]] internal method of *predicate* with *T* as *thisArgument* and a List containing *kValue*, *k*, and *O* as *argumentsList*.
      4. ReturnIfAbrupt(*testResult*).
      5. If ToBoolean(*testResult*) is **true**, return *k*.
   5. Increase *k* by 1.
10. Return -1.

The length property of the **findIndex** method is **1**.

NOTE The **findIndex** function is intentionally generic; it does not require that its **this** value be an Array object. Therefore it can be transferred to other kinds of objects for use as a method. Whether the **findIndex** function can be applied successfully to an exotic object that is not an Array is implementation-dependent.