```
type Item = (<A as Iterator>::Item, <B as Iterator>::Item)
```

The type of the elements being iterated over.

```
fn next(&mut self) -> Option<<Zip<A, B> as Iterator>::Item>
```

Advances the iterator and returns the next value. Read more

```
fn size_hint(&self) -> (usize, Option<usize>)
```

Returns the bounds on the remaining length of the iterator. Read more

```
fn nth(&mut self, n: usize) -> Option<<Zip<A, B> as Iterator>::Item>
```

Returns the `n`th element of the iterator. Read more

```
fn fold<Acc, F>(self, init: Acc, f: F) -> Acc
where
    F: FnMut(Acc, <Zip<A, B> as Iterator>::Item) -> Acc,
```

Folds every element into an accumulator by applying an operation, returning the final result. Read more

```
fn next_chunk<const N: usize>(
    &mut self
) -> Result<[Self::Item; N], IntoIter<Self::Item, N>>
where
    Self: Sized,
```

This is a nightly-only experimental API. (`iter_next_chunk` #98326)

Advances the iterator and returns an array containing the next `N` values. Read more

```
fn count(self) -> usize
where
    Self: Sized,
```

Consumes the iterator, counting the number of iterations and returning it. Read more

```
fn last(self) -> Option<Self::Item>
where
    Self: Sized,
```

Consumes the iterator, returning the last element. Read more

```
fn advance_by(&mut self, n: usize) -> Result<(), NonZeroUsize>
```

This is a nightly-only experimental API. (`iter_advance_by` #77404)

Advances the iterator by `n` elements. Read more

```
fn step_by(self, step: usize) -> StepBy<Self> ⓘ
```

1.28.0 ·

```
where
    Self: Sized,
```

Creates an iterator starting at the same point, but stepping by the given amount at each iteration. Read more

```
fn chain<U>(self, other: U) -> Chain<Self, <U as
IntoIterator>::IntoIter> ⓘ
where
    Self: Sized,
    U: IntoIterator<Item = Self::Item>,
```

Takes two iterators and creates a new iterator over both in sequence. Read more

```
fn zip<U>(self, other: U) -> Zip<Self, <U as IntoIterator>::IntoIter>
ⓘ
where
    Self: Sized,
    U: IntoIterator,
```

'Zips up' two iterators into a single iterator of pairs. Read more

```
fn intersperse_with<G>(self, separator: G) -> IntersperseWith<Self, G>
ⓘ
where
    Self: Sized,
    G: FnMut() -> Self::Item,
```

 This is a nightly-only experimental API. (`iter_intersperse` #79524)

Creates a new iterator which places an item generated by `separator` between adjacent items of the original iterator. Read more

```
fn map<B, F>(self, f: F) -> Map<Self, F> ⓘ
where
    Self: Sized,
    F: FnMut(Self::Item) -> B,
```

Takes a closure and creates an iterator which calls that closure on each element. Read more

```
fn for_each<F>(self, f: F)                                    1.21.0 ·
where
    Self: Sized,
    F: FnMut(Self::Item),
```

Calls a closure on each element of an iterator. Read more

```
fn filter<P>(self, predicate: P) -> Filter<Self, P> ⓘ
where
    Self: Sized,
    P: FnMut(&Self::Item) -> bool,
```

Creates an iterator which uses a closure to determine if an element should be yielded. Read more

```
fn filter_map<B, F>(self, f: F) -> FilterMap<Self, F> ⓘ
where
    Self: Sized,
    F: FnMut(Self::Item) -> Option<B>,
```

Creates an iterator that both filters and maps. Read more

```
fn enumerate(self) -> Enumerate<Self> ⓘ
where
    Self: Sized,
```

Creates an iterator which gives the current iteration count as well as the next value. Read more

```
fn peekable(self) -> Peekable<Self> ⓘ
where
    Self: Sized,
```

Creates an iterator which can use the `peek` and `peek_mut` methods to look at the next element of the iterator without consuming it. See their documentation for more information. Read more

```
fn skip_while<P>(self, predicate: P) -> SkipWhile<Self, P> ⓘ
where
    Self: Sized,
    P: FnMut(&Self::Item) -> bool,
```

Creates an iterator that `skip`s elements based on a predicate. Read more

```
fn take_while<P>(self, predicate: P) -> TakeWhile<Self, P> ⓘ
where
    Self: Sized,
    P: FnMut(&Self::Item) -> bool,
```

Creates an iterator that yields elements based on a predicate. Read more

```
fn map_while<B, P>(self, predicate: P) -> MapWhile<Self, P> ⓘ        1.57.0 ·
where
    Self: Sized,
    P: FnMut(Self::Item) -> Option<B>,
```

Creates an iterator that both yields elements based on a predicate and maps. Read more

```
fn skip(self, n: usize) -> Skip<Self> ⓘ
where
    Self: Sized,
```

Creates an iterator that skips the first `n` elements. Read more

```
fn take(self, n: usize) -> Take<Self> ⓘ
where
    Self: Sized,
```