Option<T>

```
// To inner type
⇒ unwrap () → T
⇒ unwrap_or (T) → T
⇒ unwrap_or_else (() → T) → T
⇒ unwrap_or_default () → T where T: Default
⇒ expect (&str) → T

// Converting to another type
⇒ map ((T) → U) → Option<U>
⇒ map_or (U, (T) → U) → U
⇒ map_or_else (() → U, (T) → U) → U

// To Result
⇒ ok_or (E) → Result<T, E>
⇒ ok_or_else (() → E) → Result<T, E>

// Conditioning
⇒ filter ((&T) → bool) → Option<T>
⇒ and (Option<U>) → Option<U>
⇒ and_then ((T) → Option<U>) → Option<U>
⇒ or (Option<T>) → Option<T>
⇒ or_else (() → Option<T>) → Option<T>
⇒ xor (Option<T>) → Option<T>
```

Option<&T>

```
// Cloning inner
⇒ cloned () → Option<T> where T: Clone
⇒ copied () → Option<T> where T: Copy
```

Option<Option<T>>

```
⇒ flatten () → Option<T>
```

Option<Result<T, E>>

```
⇒ transpose () → Result<Option<T>, E>
```

&Option<T>

```
// Checking inner
⇒ is_some () → bool
⇒ is_none () → bool

// To inner reference
⇒ as_ref () → Option<&T>
⇒ iter () → Iterator<&T>
⇒ as_deref () → Option<&U>
   where T: Deref<Target = U>
```

&mut Option<T>

```
// To inner mutable reference
⇒ as_mut () → Option<&mut T>
⇒ iter_mut () → Iterator<&mut T>
⇒ as_deref_mut () → Option<&mut U>
   where T: DerefMut + Deref<Target = U>
```

Iterator<Item = T>

```
// Mapping and filtering
⇒ map        (( T) → U)         → Iterator<Item = U>
⇒ filter     ((&T) → bool)      → Iterator<Item = T>
⇒ filter_map (( T) → Option<U>) → Iterator<Item = U>

// Collecting and folding
⇒ fold (S, (S, T) → S) → S
⇒ collect () → B where B: FromIterator<T>
⇒ partition ((&T) → bool) → (B, B) where B: Default + Extend<T>

// Counting and enumerating
⇒ count () → usize
⇒ last () → Option<T>
⇒ enumerate () → Iterator<Item = (usize, T)>

// Combining with other iterators
⇒ zip   (IntoIterator<Item = U>) → Iterator<Item = (T, U)>
⇒ chain (IntoIterator<Item = T>) → Iterator<Item = T>

// Flattening
⇒ flatten () → Iterator<U> where T: IntoIterator<U>
⇒ flat_map ((T) → IntoIterator<Item = U>) → Iterator<Item = U>

// Taking and skipping
⇒ skip (usize) → Iterator<Item = T>
⇒ take (usize) → Iterator<Item = T>
⇒ skip_while ((&T) → bool) → Iterator<Item = T>
⇒ take_while ((&T) → bool) → Iterator<Item = T>
⇒ step_by (usize) → Iterator<Item = T>

// Misc. iterating
⇒ for_each ((T) → ()) → ()
⇒ inspect ((&T) → ()) → Iterator<Item = T>
⇒ scan (S, (&mut S, T) → Option<U>) → Iterator<Item = U>

// Calculations
⇒ sum     () → S where S: Sum<T>
⇒ product () → P where P: Product<T>

// Maximum and minimum
⇒ max () → Option<T> where T: Ord
⇒ min () → Option<T> where T: Ord
⇒ max_by ((&T, &T) → Ordering) → Option<T>
⇒ min_by ((&T, &T) → Ordering) → Option<T>
⇒ max_by_key ((&T) → U) → Option<T> where U: Ord
⇒ min_by_key ((&T) → U) → Option<T> where U: Ord

// Comparing with another iterator
⇒ eq (IntoIterator<Item = T>) → bool where T: PartialEq
⇒ ne (IntoIterator<Item = T>) → bool where T: PartialEq
⇒ lt (IntoIterator<Item = T>) → bool where T: PartialOrd
⇒ le (IntoIterator<Item = T>) → bool where T: PartialOrd
⇒ gt (IntoIterator<Item = T>) → bool where T: PartialOrd
⇒ ge (IntoIterator<Item = T>) → bool where T: PartialOrd
⇒ cmp (IntoIterator<Item = T>) → Ordering where T: Ord
⇒ partial_cmp (IntoIterator<Item = T>)
   → Option<Ordering> where T: PartialOrd

// Reversing and cycling
⇒ rev   () → Iterator<Item = T> where Self: DoubleEndedIterator
⇒ cycle () → Iterator<Item = T> where Self: Clone
```

&[T]

```
// Splitting to iterator
⇒ split  ((&T) → bool) → Iterator<Item = &[T]>
⇒ rsplit ((&T) → bool) → Iterator<Item = &[T]>
⇒ splitn  (usize, (&T) → bool) → Iterator<Item = &[T]>
⇒ rsplitn (usize, (&T) → bool) → Iterator<Item = &[T]>

// Splitting at position
⇒ split_at (usize) → (&[T], &[T])
⇒ split_first () → Option<(&T, &[T])>
⇒ split_last  () → Option<(&T, &[T])>

// Chunks and windows
⇒ chunks        (usize) → Iterator<Item = &[T]>
⇒ chunks_exact  (usize) → Iterator<Item = &[T]>
⇒ rchunks       (usize) → Iterator<Item = &[T]>
⇒ rchunks_exact (usize) → Iterator<Item = &[T]>
⇒ windows       (usize) → Iterator<Item = &[T]>

// Matching
⇒ contains    (&T)   → bool where T: PartialEq
⇒ starts_with (&[T]) → bool where T: PartialEq
⇒ ends_with   (&[T]) → bool where T: PartialEq

// Binary searching
⇒ binary_search (&T)                       → Result<usize, usize> where T: Ord
⇒ binary_search_by ((&T) → Ordering)       → Result<usize, usize>
⇒ binary_search_by_key (&B, (&T) → B) → Result<usize, usize> where B: Ord

// Getting and iterating
⇒ first () → Option<&T>
⇒ last  () → Option<&T>
⇒ get (SliceIndex<[T]>) → Option<&T>
⇒ iter () → Iterator<Item = &T>

// Length
⇒ len () → usize
⇒ is_empty () → bool
```

&mut [T]

```
// Splitting to iterator
⇒ split_mut  ((&T) → bool) → Iterator<Item = &mut [T]>
⇒ rsplit_mut ((&T) → bool) → Iterator<Item = &mut [T]>
⇒ splitn_mut  (usize, (&T) → bool) → Iterator<Item = &mut [T]>
⇒ rsplitn_mut (usize, (&T) → bool) → Iterator<Item = &mut [T]>

// Splitting at position
⇒ split_at_mut (usize) → (&mut [T], &mut [T])
⇒ split_first_mut () → Option<(&mut T, &mut [T])>
⇒ split_last_mut  () → Option<(&mut T, &mut [T])>

// Chunks
⇒ chunks_mut        (usize) → Iterator<Item = &mut [T]>
⇒ chunks_exact_mut  (usize) → Iterator<Item = &mut [T]>
⇒ rchunks_mut       (usize) → Iterator<Item = &mut [T]>
⇒ rchunks_exact_mut (usize) → Iterator<Item = &mut [T]>

// Sorting
⇒ sort () where T: Ord
⇒ sort_by ((&T, &T) → Ordering)
```

&[u8]

```
// ASCII
⇒ is_asci
⇒ eq_igno
⇒ to_asci
⇒ to_asci
```

&mut

```
// ASCII
⇒ make_as
⇒ make_as
```

str

```
// Bytes
:: from_ut
:: from_ut
```

&str

```
// Chars
⇒ chars (
⇒ char_in
⇒ is_char_

// Bytes
⇒ bytes (
⇒ as_byte

// Splitti
⇒ split_a

// Splitti
⇒ lines (
⇒ split_w
⇒ split_a
⇒ split
⇒ rsplit
⇒ splitn
⇒ rsplitn
⇒ split_t
⇒ rsplit_

// Trimmin
⇒ trim
⇒ trim_st
⇒ trim_en
⇒ trim_ma
⇒ trim_st
⇒ trim_en

// Matchin
⇒ contain
⇒ starts_
⇒ ends_wi
⇒ find  (
⇒ rfind (
⇒ matches
```

Dark Single Large GitHub

```
// Mutation
⇒ take () → Option<T>
⇒ replace (T) → Option<T>
⇒ insert (T) → &mut T
⇒ get_or_insert (T) → &mut T
⇒ get_or_insert_with (() → T) → &mut T
```

Result<T, E>

```
// To inner type
⇒ unwrap () → T where E: Debug
⇒ unwrap_err () → E where T: Debug
⇒ unwrap_or (T) → T
⇒ unwrap_or_else ((E) → T) → T
⇒ unwrap_or_default () → T where T: Default
⇒ expect (&str) → T
⇒ expect_err (&str) → E
⇒ ok () → Option<T>
⇒ err () → Option<E>

// Mapping
⇒ map ((T) → U) → Result<U, E>
⇒ map_err ((E) → F) → Result<T, F>
⇒ map_or (U, (T) → U) → U
⇒ map_or_else ((E) → U, (T) → U) → U

// Conditioning
⇒ and (Result<U, E>) → Result<U, E>
⇒ and_then ((T) → Result<U, E>) → Result<U, E>
⇒ or (Result<T, F>) → Result<T, F>
⇒ or_else ((E) → Result<T, F>) → Result<T, F>
```

Result<Option<T>, E>

```
// Transposing
⇒ transpose () → Option<Result<T, E>>
```

&Result<T, E>

```
// Checking inner
⇒ is_ok () → bool
⇒ is_err () → bool

// To inner reference
⇒ as_ref () → Result<&T, &E>
⇒ iter () → Iterator<Item = &T>
```

&mut Result<T, E>

```
// To inner mutable reference
⇒ as_mut () → Result<&mut T, &mut E>
⇒ iter_mut () → Iterator<Item = &mut T>
```

Iterator<Item = &T>

```
// Cloning inner
⇒ cloned () → Iterator<T> where T: Clone
⇒ copied () → Iterator<T> where T: Copy
```

&mut Iterator<Item = T>

```
// Finding and positioning
⇒ find      ((&T) → bool)      → Option<T>
⇒ find_map  (( T) → Option<U>) → Option<U>
⇒ position  (( T) → bool)      → Option<usize>
⇒ rposition (( T) → bool)      → Option<usize>
   where Self: ExactSizeIterator + DoubleEndedIterator

// Boolean operations
⇒ all ((T) → bool) → bool
⇒ any ((T) → bool) → bool

// Try iterating
⇒ try_for_each   ((T) → R) → R where R: Try<Ok = ()>
⇒ try_fold (S, (S, T) → R) → R where R: Try<Ok = S>
```

iter

```
// Creating simple iterators
:: empty () → Iterator<Item = T>
:: once (T) → Iterator<Item = T>
:: once_with (() → T) → Iterator<Item = T>
:: repeat (T) → Iterator<Item = T> where T: Clone
:: repeat_with (() → T) → Iterator<Item = T>
:: from_fn (() → Option<T>) → Iterator<Item = T>
:: successors (Option<T>, (&T) → Option<T>) → Iterator<Item = T>
```

```
⇒ sort_by_key ((&T) → K) where K: Ord
⇒ sort_by_cached_key ((&T) → K) where K: Ord
⇒ sort_unstable () where T: Ord
⇒ sort_unstable_by ((&T, &T) → Ordering)
⇒ sort_unstable_by_key ((&T) → K) where K: Ord

// Rearranging
⇒ swap (usize, usize)
⇒ reverse ()
⇒ rotate_left (usize)
⇒ rotate_right (usize)

// Overriding
⇒ swap_with_slice  (&mut [T])
⇒ copy_from_slice  (&[T]) where T: Copy
⇒ clone_from_slice (&[T]) where T: Clone

// Getting and iterating
⇒ first_mut () → Option<&mut T>
⇒ last_mut  () → Option<&mut T>
⇒ get_mut (SliceIndex<[T]>) → Option<&mut T>
⇒ iter_mut () → Iterator<Item = &mut T>
```

&mut Vec<T>

```
// Adding and removing single item
⇒ push (T)
⇒ pop () → Option<T>
⇒ insert (usize, T)
⇒ remove (usize) → T
⇒ swap_remove (usize) → T

// Extending
⇒ append (&mut Vec<T>)
⇒ extend (IntoIterator<Item = T>)
⇒ extend (IntoIterator<Item = &T>) where T: Copy
⇒ extend_from_slice (&[T]) where T: Clone

// Resizing
⇒ truncate (usize)
⇒ resize (usize, T) where T: Clone
⇒ resize_with (usize, () → T)

// Clearing
⇒ clear ()
⇒ retain ((&T) → bool)

// Removing or replacing range into iterator
⇒ drain  (RangeBounds<usize>) → Iterator<T>
⇒ splice (RangeBounds<usize>, IntoIterator<Item = T>) → Iterator<T>

// Deduplicating
⇒ dedup () where T: PartialEq
⇒ dedup_by ((&mut T, &mut T) → bool)
⇒ dedup_by_key ((&mut T) → K) where K: PartialEq

// Splitting off
⇒ split_off (usize) → Vec<T>

// Capacity manipulation
⇒ reserve (usize)
⇒ reserve_exact (usize)
⇒ shrink_to_fit ()
```

```
⇒ rmatche
⇒ match_i
⇒ rmatch_

// Case
⇒ to_uppe
⇒ to_lowe
⇒ to_asci
⇒ to_asci
⇒ eq_igno

// Replaci
⇒ replace
⇒ replace

// Length
⇒ len ()
⇒ is_empt

// Misc.
⇒ is_asci
⇒ repeat
⇒ encode_
⇒ parse (
```

&mut

```
// Splitti
⇒ split_a

// Case co
⇒ make_as
⇒ make_as
```

&mut

```
// Inserti
⇒ push_st
⇒ insert_

// Adding
⇒ push (c
⇒ pop ()
⇒ insert
⇒ remove

// Clearin
⇒ clear (
⇒ truncat
⇒ retain

// Capacit
⇒ reserve
⇒ reserve_
⇒ shrink_

// Misc.
⇒ split_o
⇒ replace_
⇒ drain (
```

Dark Single Large GitHub

slice

```
// Creating slice from reference
:: from_ref (&T) → &[T]
:: from_mut (&mut T) → &mut [T]
```

Dark Single Large GitHub