

- `aggregate()` — see [reduce\(\)](#)
- `all()` — determine whether all items emitted by an Observable meet some criteria
- `amb()` — given two or more source Observables, emits all of the items from the first of these Observables to emit an item
- `ambWith()` — instance version of [amb\(\)](#)
- `and()` — combine the emissions from two or more source Observables into a `Pattern (rxjava-joins)`
- `apply()` (scala) — see [create\(\)](#)
- `asObservable()` (kotlin) — see [from\(\)](#) (et al.)
- `asyncAction()` — convert an Action into an Observable that executes the Action and emits its return value (`rxjava-async`)
- `asyncFunc()` — convert a function into an Observable that executes the function and emits its return value (`rxjava-async`)
- `averageDouble()` — calculates the average of Doubles emitted by an Observable and emits this average (`rxjava-math`)
- `averageFloat()` — calculates the average of Floats emitted by an Observable and emits this average (`rxjava-math`)
- `averageInteger()` — calculates the average of Integers emitted by an Observable and emits this average (`rxjava-math`)
- `averageLong()` — calculates the average of Longs emitted by an Observable and emits this average (`rxjava-math`)
- `blocking()` (clojure) — see [toBlocking\(\)](#)
- `buffer()` — periodically gather items from an Observable into bundles and emit these bundles rather than emitting the items one at a time
- `byLine()` (`StringObservable`) — converts an Observable of Strings into an Observable of Lines by treating the source sequence as a stream and splitting it on line-endings
- `cache()` — remember the sequence of items emitted by the Observable and emit the same sequence to future Subscribers
- `cast()` — cast all items from the source Observable into a particular type before reemitting them
- `catch()` (clojure) — see [onErrorResumeNext\(\)](#)
- `chunkify()` — returns an iterable that periodically returns a list of items emitted by the source Observable since the last list (??)
- `collect()` — collects items emitted by the source Observable into a single mutable data structure and returns an Observable that emits this structure
- `combineLatest()` — when an item is emitted by either of two Observables, combine the latest item emitted by each Observable via a specified function and emit items based on the results of this function
- `combineLatestWith()` (scala) — instance version of [combineLatest\(\)](#)
- `concat()` — concatenate two or more Observables sequentially
- `concatMap()` — transform the items emitted by an Observable into Observables, then flatten this into a single Observable, without interleaving
- `concatWith()` — instance version of [concat\(\)](#)
- `connect()` — instructs a Connectable Observable to begin emitting items
- `cons()` (clojure) — see [concat\(\)](#)
- `contains()` — determine whether an Observable emits a particular item or not
- `count()` — counts the number of items emitted by an Observable and emits this count
- `countLong()` — counts the number of items emitted by an Observable and emits this count
- `create()` — create an Observable from scratch by means of a function

- `cycle()` (closure) — see [repeat\(\)](#)
- [debounce\(\)](#) — only emit an item from the source Observable after a particular timespan has passed without the Observable emitting any other items
- [decode\(\)](#) (`StringObservable`) — convert a stream of multibyte characters into an Observable that emits byte arrays that respect character boundaries
- [defaultIfEmpty\(\)](#) — emit items from the source Observable, or emit a default item if the source Observable completes after emitting no items
- [defer\(\)](#) — do not create the Observable until a Subscriber subscribes; create a fresh Observable on each subscription
- [deferFuture\(\)](#) — convert a Future that returns an Observable into an Observable, but do not attempt to get the Observable that the Future returns until a Subscriber subscribes (`rxjava-async`)
- [deferCancellableFuture\(\)](#) — convert a Future that returns an Observable into an Observable in a way that monitors the subscription status of the Observable to determine whether to halt work on the Future, but do not attempt to get the returned Observable until a Subscriber subscribes (??)(`rxjava-async`)
- [delay\(\)](#) — shift the emissions from an Observable forward in time by a specified amount
- [dematerialize\(\)](#) — convert a materialized Observable back into its non-materialized form
- [distinct\(\)](#) — suppress duplicate items emitted by the source Observable
- [distinctUntilChanged\(\)](#) — suppress duplicate consecutive items emitted by the source Observable
- `do()` (closure) — see [doOnEach\(\)](#)
- [doOnCompleted\(\)](#) — register an action to take when an Observable completes successfully
- [doOnEach\(\)](#) — register an action to take whenever an Observable emits an item
- [doOnError\(\)](#) — register an action to take when an Observable completes with an error
- `doOnNext()` — see [doOnEach\(\)](#)
- `doOnRequest()` — register an action to take when items are requested from an Observable via reactive-pull backpressure (??)
- [doOnSubscribe\(\)](#) — register an action to take when an observer subscribes to an Observable
- [doOnTerminate\(\)](#) — register an action to take when an Observable completes, either successfully or with an error
- [doOnUnsubscribe\(\)](#) — register an action to take when an observer unsubscribes from an Observable
- [doWhile\(\)](#) — emit the source Observable's sequence, and then repeat the sequence as long as a condition remains true (`contrib-computation-expressions`)
- `drop()` (scala/closure) — see [skip\(\)](#)
- `dropRight()` (scala) — see [skipLast\(\)](#)
- `dropUntil()` (scala) — see [skipUntil\(\)](#)
- `dropWhile()` (scala) — see [skipWhile\(\)](#)
- `drop-while()` (closure) — see [skipWhile\(\)](#)
- [elementAt\(\)](#) — emit item *n* emitted by the source Observable
- [elementAtOrDefault\(\)](#) — emit item *n* emitted by the source Observable, or a default item if the source Observable emits fewer than *n* items
- [empty\(\)](#) — create an Observable that emits nothing and then completes
- [encode\(\)](#) (`StringObservable`) — transform an Observable that emits strings into an Observable that emits byte arrays that respect character boundaries of multibyte characters in the original strings
- [error\(\)](#) — create an Observable that emits nothing and then signals an error
- `every()` (closure) — see [all\(\)](#)
- [exists\(\)](#) — determine whether an Observable emits any items or not
- [filter\(\)](#) — filter items emitted by an Observable

- `finally()` (closure) — see [finallyDo\(\)](#)
- `filterNot()` (scala) — see [filter\(\)](#)
- [finallyDo\(\)](#) — register an action to take when an Observable completes
- [first\(\)](#) (Observable) — emit only the first item emitted by an Observable, or the first item that meets some condition
- [first\(\)](#) (BlockingObservable) — emit only the first item emitted by an Observable, or the first item that meets some condition
- [firstOrDefault\(\)](#) (Observable) — emit only the first item emitted by an Observable, or the first item that meets some condition, or a default value if the source Observable is empty
- [firstOrDefault\(\)](#) (BlockingObservable) — emit only the first item emitted by an Observable, or the first item that meets some condition, or a default value if the source Observable is empty
- `firstOrElse()` (scala) — see [firstOrDefault\(\)](#) or [firstOrDefault\(\)](#) (BlockingObservable)
- [flatMap\(\)](#) — transform the items emitted by an Observable into Observables, then flatten this into a single Observable
- [flatMapIterable\(\)](#) — create Iterables corresponding to each emission from a source Observable and merge the results into a single Observable
- `flatMapIterableWith()` (scala) — instance version of [flatMapIterable\(\)](#)
- `flatMapWith()` (scala) — instance version of [flatMap\(\)](#)
- `flatten()` (scala) — see [merge\(\)](#)
- `flattenDelayError()` (scala) — see [mergeDelayError\(\)](#)
- `foldLeft()` (scala) — see [reduce\(\)](#)
- `forall()` (scala) — see [all\(\)](#)
- `forEach()` (Observable) — see [subscribe\(\)](#)
- [forEach\(\)](#) (BlockingObservable) — invoke a function on each item emitted by the Observable; block until the Observable completes
- [forEachFuture\(\)](#) (Async) — pass Subscriber methods to an Observable but also have it behave like a Future that blocks until it completes (rxjava-async)
- [forEachFuture\(\)](#) (BlockingObservable) — create a futureTask that will invoke a specified function on each item emitted by an Observable (??)
- [forIterable\(\)](#) — apply a function to the elements of an Iterable to create Observables which are then concatenated (??)
- [from\(\)](#) — convert an Iterable, a Future, or an Array into an Observable
- [from\(\)](#) (StringObservable) — convert a stream of characters or a Reader into an Observable that emits byte arrays or Strings
- [fromAction\(\)](#) — convert an Action into an Observable that invokes the action and emits its result when a Subscriber subscribes (rxjava-async)
- [fromCallable\(\)](#) — convert a Callable into an Observable that invokes the callable and emits its result or exception when a Subscriber subscribes (rxjava-async)
- [fromCancellableFuture\(\)](#) — convert a Future into an Observable in a way that monitors the subscription status of the Observable to determine whether to halt work on the Future, but do not attempt to get the Future's value until a Subscriber subscribes (??)(rxjava-async)
- `fromFunc0()` — see [fromCallable\(\)](#) (rxjava-async)
- [fromFuture\(\)](#) — convert a Future into an Observable, but do not attempt to get the Future's value until a Subscriber subscribes (??)
- [fromRunnable\(\)](#) — convert a Runnable into an Observable that invokes the runnable and emits its result when a Subscriber subscribes (rxjava-async)

- [generate\(_\)](#) — create an Observable that emits a sequence of items as generated by a function of your choosing (??)
- [generateAbsoluteTime\(_\)](#) — create an Observable that emits a sequence of items as generated by a function of your choosing, with each item emitted at an item-specific time (??)
- [generator\(\)](#) (clojure) — see [generate\(_\)](#)
- [getIterator\(_\)](#) — convert the sequence emitted by the Observable into an Iterator
- [groupBy\(_\)](#) — divide an Observable into a set of Observables that emit groups of items from the original Observable, organized by key
- [group-by\(\)](#) (clojure) — see [groupBy\(_\)](#)
- [groupByUntil\(_\)](#) — a variant of the [groupBy\(_\)](#) operator that closes any open GroupedObservable upon a signal from another Observable (??)
- [groupJoin\(_\)](#) — combine the items emitted by two Observables whenever one item from one Observable falls within a window of duration specified by an item emitted by the other Observable
- [head\(\)](#) (scala) — see [first\(_\)](#) (*BlockingObservable*)
- [headOption\(\)](#) (scala) — see [firstOrDefault\(_\)](#) or [firstOrDefault\(_\)](#) (*BlockingObservable*)
- [headOrElse\(\)](#) (scala) — see [firstOrDefault\(_\)](#) or [firstOrDefault\(_\)](#) (*BlockingObservable*)
- [ifThen\(_\)](#) — only emit the source Observable's sequence if a condition is true, otherwise emit an empty or default sequence (*contrib-computation-expressions*)
- [ignoreElements\(_\)](#) — discard the items emitted by the source Observable and only pass through the error or completed notification
- [interval\(_\)](#) — create an Observable that emits a sequence of integers spaced by a given time interval
- [into\(\)](#) (clojure) — see [reduce\(_\)](#)
- [isEmpty\(_\)](#) — determine whether an Observable emits any items or not
- [items\(\)](#) (scala) — see [just\(_\)](#)
- [join\(_\)](#) — combine the items emitted by two Observables whenever one item from one Observable falls within a window of duration specified by an item emitted by the other Observable
- [join\(_\)](#) (*StringObservable*) — converts an Observable that emits a sequence of strings into an Observable that emits a single string that concatenates them all, separating them by a specified string
- [just\(_\)](#) — convert an object into an Observable that emits that object
- [last\(_\)](#) (*BlockingObservable*) — block until the Observable completes, then return the last item emitted by the Observable
- [last\(_\)](#) (*Observable*) — emit only the last item emitted by the source Observable
- [lastOption\(\)](#) (scala) — see [lastOrDefault\(_\)](#) or [lastOrDefault\(_\)](#) (*BlockingObservable*)
- [lastOrDefault\(_\)](#) (*BlockingObservable*) — block until the Observable completes, then return the last item emitted by the Observable or a default item if there is no last item
- [lastOrDefault\(_\)](#) (*Observable*) — emit only the last item emitted by an Observable, or a default value if the source Observable is empty
- [lastOrElse\(\)](#) (scala) — see [lastOrDefault\(_\)](#) or [lastOrDefault\(_\)](#) (*BlockingObservable*)
- [latest\(_\)](#) — returns an iterable that blocks until or unless the Observable emits an item that has not been returned by the iterable, then returns the latest such item
- [length\(\)](#) (scala) — see [count\(_\)](#)
- [limit\(\)](#) — see [take\(_\)](#)
- [longCount\(\)](#) (scala) — see [countLong\(_\)](#)

- [map\(\)](#) — transform the items emitted by an Observable by applying a function to each of them
- [mapcat\(\)](#) (closure) — see [concatMap\(\)](#)
- [mapMany\(\)](#) — see: [flatMap\(\)](#)
- [materialize\(\)](#) — convert an Observable into a list of Notifications
- [max\(\)](#) — emits the maximum value emitted by a source Observable (`rxjava-math`)
- [maxBy\(\)](#) — emits the item emitted by the source Observable that has the maximum key value (`rxjava-math`)
- [merge\(\)](#) — combine multiple Observables into one
- [mergeDelayError\(\)](#) — combine multiple Observables into one, allowing error-free Observables to continue before propagating errors
- [merge-delay-error\(\)](#) (closure) — see [mergeDelayError\(\)](#)
- [mergeMap\(\)](#) * — see: [flatMap\(\)](#)
- [mergeMapIterable\(\)](#) — see: [flatMapIterable\(\)](#)
- [mergeWith\(\)](#) — instance version of [merge\(\)](#)
- [min\(\)](#) — emits the minimum value emitted by a source Observable (`rxjava-math`)
- [minBy\(\)](#) — emits the item emitted by the source Observable that has the minimum key value (`rxjava-math`)
- [mostRecent\(\)](#) — returns an iterable that always returns the item most recently emitted by the Observable
- [multicast\(\)](#) — represents an Observable as a Connectable Observable
- [never\(\)](#) — create an Observable that emits nothing at all
- [next\(\)](#) — returns an iterable that blocks until the Observable emits another item, then returns that item
- [notEmpty\(\)](#) (scala) — see [isEmpty\(\)](#)
- [nth\(\)](#) (closure) — see [elementAt\(\)](#) and [elementAtOrDefault\(\)](#)
- [observeOn\(\)](#) — specify on which Scheduler a Subscriber should observe the Observable
- [ofType\(\)](#) — emit only those items from the source Observable that are of a particular class
- [onBackpressureBlock\(\)](#) — block the Observable's thread until the Observer is ready to accept more items from the Observable (??)
- [onBackpressureBuffer\(\)](#) — maintain a buffer of all emissions from the source Observable and emit them to downstream Subscribers according to the requests they generate
- [onBackpressureDrop\(\)](#) — drop emissions from the source Observable unless there is a pending request from a downstream Subscriber, in which case emit enough items to fulfill the request
- [onErrorFlatMap\(\)](#) — instructs an Observable to emit a sequence of items whenever it encounters an error (??)
- [onErrorResumeNext\(\)](#) — instructs an Observable to emit a sequence of items if it encounters an error
- [onErrorReturn\(\)](#) — instructs an Observable to emit a particular item when it encounters an error
- [onExceptionResumeNext\(\)](#) — instructs an Observable to continue emitting items after it encounters an exception (but not another variety of throwable)
- [orElse\(\)](#) (scala) — see [defaultIfEmpty\(\)](#)
- [parallel\(\)](#) — split the work done on the emissions from an Observable into multiple Observables each operating on its own parallel thread (??)
- [parallelMerge\(\)](#) — combine multiple Observables into smaller number of Observables (??)
- [pivot\(\)](#) — combine multiple sets of grouped observables so that they are arranged primarily by group rather than by set (??)
- [publish\(\)](#) — represents an Observable as a Connectable Observable
- [publishLast\(\)](#) — represent an Observable as a Connectable Observable that emits only the last item emitted by the source Observable (??)

- [`range\(\)`](#) — create an Observable that emits a range of sequential integers
- [`reduce\(\)`](#) — apply a function to each emitted item, sequentially, and emit only the final accumulated value
- [`reductions\(\)`](#) (closure) — see [`scan\(\)`](#)
- [`refCount\(\)`](#) — makes a Connectable Observable behave like an ordinary Observable
- [`repeat\(\)`](#) — create an Observable that emits a particular item or sequence of items repeatedly
- [`repeatWhen\(\)`](#) — create an Observable that emits a particular item or sequence of items repeatedly, depending on the emissions of a second Observable
- [`replay\(\)`](#) — ensures that all Subscribers see the same sequence of emitted items, even if they subscribe after the Observable begins emitting the items
- [`rest\(\)`](#) (closure) — see [`next\(\)`](#)
- [`return\(\)`](#) (closure) — see [`just\(\)`](#)
- [`retry\(\)`](#) — if a source Observable emits an error, resubscribe to it in the hopes that it will complete without error
- [`retryWhen\(\)`](#) — if a source Observable emits an error, pass that error to another Observable to determine whether to resubscribe to the source
- [`runAsync\(\)`](#) — returns a `StoppableObservable` that emits multiple actions as generated by a specified Action on a Scheduler (`rxjava-async`)
- [`sample\(\)`](#) — emit the most recent items emitted by an Observable within periodic time intervals
- [`scan\(\)`](#) — apply a function to each item emitted by an Observable, sequentially, and emit each successive value
- [`seq\(\)`](#) (closure) — see [`getIterator\(\)`](#)
- [`sequenceEqual\(\)`](#) — test the equality of sequences emitted by two Observables
- [`sequenceEqualWith\(\)`](#) (scala) — instance version of [`sequenceEqual\(\)`](#)
- [`serialize\(\)`](#) — force an Observable to make serialized calls and to be well-behaved
- [`share\(\)`](#) — see [`refCount\(\)`](#)
- [`single\(\)`](#) (`BlockingObservable`) — if the source Observable completes after emitting a single item, return that item, otherwise throw an exception
- [`single\(\)`](#) (`Observable`) — if the source Observable completes after emitting a single item, emit that item, otherwise notify of an exception
- [`singleOption\(\)`](#) (scala) — see [`singleOrDefault\(\)`](#) (`BlockingObservable`)
- [`singleOrDefault\(\)`](#) (`BlockingObservable`) — if the source Observable completes after emitting a single item, return that item, otherwise return a default item
- [`singleOrDefault\(\)`](#) (`Observable`) — if the source Observable completes after emitting a single item, emit that item, otherwise emit a default item
- [`singleOrElse\(\)`](#) (scala) — see [`singleOrDefault\(\)`](#)
- [`size\(\)`](#) (scala) — see [`count\(\)`](#)
- [`skip\(\)`](#) — ignore the first n items emitted by an Observable
- [`skipLast\(\)`](#) — ignore the last n items emitted by an Observable
- [`skipUntil\(\)`](#) — discard items emitted by a source Observable until a second Observable emits an item, then emit the remainder of the source Observable's items
- [`skipWhile\(\)`](#) — discard items emitted by an Observable until a specified condition is false, then emit the remainder
- [`sliding\(\)`](#) (scala) — see [`window\(\)`](#)
- [`slidingBuffer\(\)`](#) (scala) — see [`buffer\(\)`](#)
- [`split\(\)`](#) (`StringObservable`) — converts an Observable of Strings into an Observable of Strings that treats the source sequence as a stream and splits it on a specified regex boundary

- [`start\(\)`](#) — create an Observable that emits the return value of a function (`rxjava-async`)
- [`startCancelableFuture\(\)`](#) — convert a function that returns Future into an Observable that emits that Future's return value in a way that monitors the subscription status of the Observable to determine whether to halt work on the Future (??)(`rxjava-async`)
- [`startFuture\(\)`](#) — convert a function that returns Future into an Observable that emits that Future's return value (`rxjava-async`)
- [`startWith\(\)`](#) — emit a specified sequence of items before beginning to emit the items from the Observable
- [`stringConcat\(\)`](#) (`StringObservable`) — converts an Observable that emits a sequence of strings into an Observable that emits a single string that concatenates them all
- [`subscribeOn\(\)`](#) — specify which Scheduler an Observable should use when its subscription is invoked
- [`sumDouble\(\)`](#) — adds the Doubles emitted by an Observable and emits this sum (`rxjava-math`)
- [`sumFloat\(\)`](#) — adds the Floats emitted by an Observable and emits this sum (`rxjava-math`)
- [`sumInt\(\)`](#) — adds the Integers emitted by an Observable and emits this sum (`rxjava-math`)
- [`sumLong\(\)`](#) — adds the Longs emitted by an Observable and emits this sum (`rxjava-math`)
- [`switch\(\)`](#) (scala) — see [`switchOnNext\(\)`](#)
- [`switchCase\(\)`](#) — emit the sequence from a particular Observable based on the results of an evaluation (`contrib-computation-expressions`)
- [`switchMap\(\)`](#) — transform the items emitted by an Observable into Observables, and mirror those items emitted by the most-recently transformed Observable
- [`switchOnNext\(\)`](#) — convert an Observable that emits Observables into a single Observable that emits the items emitted by the most-recently emitted of those Observables
- [`synchronize\(\)`](#) — see [`serialize\(\)`](#)
- [`take\(\)`](#) — emit only the first n items emitted by an Observable
- [`takeFirst\(\)`](#) — emit only the first item emitted by an Observable, or the first item that meets some condition
- [`takeLast\(\)`](#) — only emit the last n items emitted by an Observable
- [`takeLastBuffer\(\)`](#) — emit the last n items emitted by an Observable, as a single list item
- [`takeRight\(\)`](#) (scala) — see [`last\(\)`](#) (`Observable`) or [`takeLast\(\)`](#)
- [`takeUntil\(\)`](#) — emits the items from the source Observable until a second Observable emits an item
- [`takeWhile\(\)`](#) — emit items emitted by an Observable as long as a specified condition is true, then skip the remainder
- [`take-while\(\)`](#) (clojure) — see [`takeWhile\(\)`](#)
- [`then\(\)`](#) — transform a series of `Pattern` objects via a `Plan` template (`rxjava-joins`)
- [`throttleFirst\(\)`](#) — emit the first items emitted by an Observable within periodic time intervals
- [`throttleLast\(\)`](#) — emit the most recent items emitted by an Observable within periodic time intervals
- [`throttleWithTimeout\(\)`](#) — only emit an item from the source Observable after a particular timespan has passed without the Observable emitting any other items
- [`throw\(\)`](#) (clojure) — see [`error\(\)`](#)
- [`timeInterval\(\)`](#) — emit the time lapsed between consecutive emissions of a source Observable
- [`timeout\(\)`](#) — emit items from a source Observable, but issue an exception if no item is emitted in a specified timespan
- [`timer\(\)`](#) — create an Observable that emits a single item after a given delay
- [`timestamp\(\)`](#) — attach a timestamp to every item emitted by an Observable
- [`toAsync\(\)`](#) — convert a function or Action into an Observable that executes the function and emits its return value (`rxjava-async`)
- [`toBlocking\(\)`](#) — transform an Observable into a BlockingObservable

- `toBlockingObservable()` - see [toBlocking\(\)](#)
- [toFuture\(\)](#) — convert the Observable into a Future
- [toIterable\(\)](#) — convert the sequence emitted by the Observable into an Iterable
- `toIterator()` — see [getIterator\(\)](#)
- [toList\(\)](#) — collect all items from an Observable and emit them as a single List
- [toMap\(\)](#) — convert the sequence of items emitted by an Observable into a map keyed by a specified key function
- [toMultimap\(\)](#) — convert the sequence of items emitted by an Observable into an ArrayList that is also a map keyed by a specified key function
- `toSeq()` (scala) — see [toList\(\)](#)
- [toSortedList\(\)](#) — collect all items from an Observable and emit them as a single, sorted List
- `tumbling()` (scala) — see [window\(\)](#)
- `tumblingBuffer()` (scala) — see [buffer\(\)](#)
- [using\(\)](#) — create a disposable resource that has the same lifespan as an Observable
- [when\(\)](#) — convert a series of `Plan` objects into an Observable (`rxjava-joins`)
- `where()` — see: [filter\(\)](#)
- [whileDo\(\)](#) — if a condition is true, emit the source Observable's sequence and then repeat the sequence as long as the condition remains true (`contrib-computation-expressions`)
- [window\(\)](#) — periodically subdivide items from an Observable into Observable windows and emit these windows rather than emitting the items one at a time
- [zip\(\)](#) — combine sets of items emitted by two or more Observables together via a specified function and emit items based on the results of this function
- `zipWith()` — *instance version of* [zip\(\)](#)
- `zipWithIndex()` (scala) — see [zip\(\)](#)
- `++` (scala) — see [concat\(\)](#)
- `+` (scala) — see [startWith\(\)](#)

(??) — this proposed operator is not part of RxJava 1.0