API Collection

Collections

Store and organize data using arrays, dictionaries, sets, and other data structures.

Topics

Arrays and Dictionaries

`struct Array`

An ordered, random-access collection.

`struct Dictionary`

A collection whose elements are key-value pairs.

`struct InlineArray`

A fixed-size array.

Sets

`struct Set`

An unordered collection of unique elements.

`protocol OptionSet`

A type that presents a mathematical set interface to a bit set.

Ranges

Create a collection of all the values in a range by using the half-open (`..<`) and closed (`...`) range operators.

`static func ..< (Self, Self) -> Range<Self>`

Returns a half-open range that contains its lower bound but not its upper bound.

`struct Range`

A half-open interval from a lower bound up to, but not including, an upper bound.

`struct RangeSet`

A set of values of any comparable type, represented by ranges.

`static func ... (Self, Self) -> ClosedRange<Self>`

Returns a closed range that contains both of its bounds.

`struct ClosedRange`

An interval from a lower bound up to, and including, an upper bound.

Strides

Create a stride that steps over values between two boundaries using the `stride(from:to:by:)` and `stride(from:through:by:)` functions.

`func stride<T>(from: T, to: T, by: T.Stride) -> StrideTo<T>`

Returns a sequence from a starting value to, but not including, an end value, stepping by the specified amount.

`func stride<T>(from: T, through: T, by: T.Stride) -> StrideThrough<T>`

Returns a sequence from a starting value toward, and possibly including, an end value, stepping by the specified amount.

Special-Use Collections

These collections can store zero, one, or many of the same element.

`func repeatElement<T>(T, count: Int) -> Repeated<T>`

Creates a collection containing the specified number of the given element.

`struct CollectionOfOne`

A collection containing a single element.

`struct EmptyCollection`

A collection whose element type is `Element` but that is always empty.

`struct KeyValuePairs`

A lightweight collection of key-value pairs.

```
typealias DictionaryLiteral
```

Dynamic Sequences

```
func sequence<T>(first: T, next: (T) -> T?) -> UnfoldFirstSequence<T>
```

Returns a sequence formed from `first` and repeated lazy applications of `next`.

```
func sequence<T, State>(state: State, next: (inout State) -> T?) ->
UnfoldSequence<T, State>
```

Returns a sequence formed from repeated lazy applications of `next` to a mutable `state`.

Joint Iteration

```
func zip<Sequence1, Sequence2>(Sequence1, Sequence2) -> Zip2Sequence<
Sequence1, Sequence2>
```

Creates a sequence of pairs built out of two underlying sequences.

Advanced Collection Topics

Sequence and Collection Protocols

Write generic code that works with any collection, or build your own collection types.

Supporting Types

Use wrappers, indices, and iterators in operations like slicing, flattening, and reversing a collection.

Managed Buffers

Build your own buffer-backed collection types.

See Also

Values and Collections

Numbers and Basic Values

Model data with numbers, Boolean values, and other fundamental types.

Strings and Text

Work with text using Unicode-safe strings.

Time

Measure how long an operation takes and determine schedules in the future.