

```
/// The type-erased, dynamic output of a  
regular expression match.
```

```
///
```

```
/// When you find a match using regular  
expression that has `AnyRegexOutput`
```

```
/// as its output type, you can find  
information about matches by iterating
```

```
@available(macOS 13.0, iOS 16.0, watchOS  
9.0, tvOS 16.0, *)
```

```
public struct AnyRegexOutput {  
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS  
9.0, tvOS 16.0, *)
```

```
extension AnyRegexOutput {
```

```
    /// Creates a dynamic regular  
expression match output from an existing  
match.
```

```
    ///
```

```
    /// You can use this initializer when  
you need an `AnyRegexOutput` instance
```

```
    /// instead of the output type of a  
strongly-typed `Regex.Match`.
```

```
    public init<Output>(_ match:  
Regex<Output>.Match)
```

```
    /// Returns strongly-typed match  
output by converting this type-erased
```

```
    /// output to the specified type, if  
possible.
```

```
    ///
```

```

    /// - Parameter outputType: The
    expected output type.
    /// - Returns: The output, if the
    underlying value can be converted to
    ///   `outputType`; otherwise, `nil`.
    public func extractValues<Output>(as
    outputType: Output.Type = Output.self) ->
    Output?
}

```

```

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension AnyRegexOutput :
RandomAccessCollection {

```

```

    /// An individual match output value.
    public struct Element {

        /// The range over which a value
        was captured, if there was a capture.
        ///
        /// If nothing was captured,
        `range` is `nil`.
        public var range:
        Range<String.Index>? { get }

        /// The slice of the input which
        was captured, if there was a capture.
        ///
        /// If nothing was captured,
        `substring` is `nil`.
        public var substring: Substring?
        { get }
    }
}

```

```
        /// The captured value, if there
was a capture.
```

```
        ///
        /// If nothing was captured,
`value` is `nil`.
        public var value: Any? { get }
```

```
        /// The type of this capture.
        public var type: any Any.Type {
get }
```

```
        /// The name of this capture, if
the capture is named.
```

```
        ///
        /// If the capture is unnamed,
`name` is `nil`.
        public var name: String? { get }
    }
```

```
        /// The position of the first element
in a nonempty collection.
```

```
        ///
        /// If the collection is empty,
`startIndex` is equal to `endIndex`.
        public var startIndex: Int { get }
```

```
        /// The collection's "past the end"
position---that is, the position one
```

```
        /// greater than the last valid
subscript argument.
```

```
        ///
        /// When you need a range that
```

includes the last element of a collection, use

```
    /// the half-open range operator  
(`..<`) with `endIndex`. The `..<`  
operator
```

```
    /// creates a range that doesn't  
include the upper bound, so it's always  
    /// safe to use with `endIndex`. For  
example:
```

```
    ///  
    ///      let numbers = [10, 20, 30,  
40, 50]  
    ///      if let index =  
numbers.firstIndex(of: 30) {  
    ///          print(numbers[index ..<  
numbers.endIndex])  
    ///      }  
    ///      // Prints "[30, 40, 50]"  
    ///
```

```
    /// If the collection is empty,  
`endIndex` is equal to `startIndex`.  
    public var endIndex: Int { get }
```

```
    /// The number of elements in the  
collection.
```

```
    ///  
    /// To check whether a collection is  
empty, use its `isEmpty` property  
    /// instead of comparing `count` to  
zero. Unless the collection guarantees  
    /// random-access performance,  
calculating `count` can be an  $O(n)$   
    /// operation.
```

```

    ///
    /// - Complexity:  $O(1)$  if the
collection conforms to
    ///   `RandomAccessCollection`;
otherwise,  $O(*n*)$ , where  $*n*$  is the
length
    ///   of the collection.
    public var count: Int { get }

    /// Returns the position immediately
after the given index.
    ///
    /// The successor of an index must be
well defined. For an index `i` into a
    /// collection `c`, calling
`c.index(after: i)` returns the same
index every
    /// time.
    ///
    /// - Parameter i: A valid index of
the collection. `i` must be less than
    ///   `endIndex`.
    /// - Returns: The index value
immediately after `i`.
    public func index(after i: Int) ->
Int

    /// Returns the position immediately
before the given index.
    ///
    /// - Parameter i: A valid index of
the collection. `i` must be greater than
    ///   `startIndex`.

```

```
    /// - Returns: The index value
    immediately before `i`.
    public func index(before i: Int) ->
    Int
```

```
    /// Accesses the element at the
    specified position.
    ///
    /// The following example accesses an
    element of an array through its
    subscript to print its value:
    ///
    ///     var streets = ["Adams",
    "Bryant", "Channing", "Douglas",
    "Evarts"]
    ///     print(streets[1])
    ///     // Prints "Bryant"
    ///
    /// You can subscript a collection
    with any valid index other than the
    /// collection's end index. The end
    index refers to the position one past
    /// the last element of a collection,
    so it doesn't correspond with an
    /// element.
    ///
    /// - Parameter position: The
    position of the element to access.
    `position`
    /// must be a valid index of the
    collection that is not equal to the
    /// `endIndex` property.
    ///
```

```

    /// - Complexity: O(1)
    public subscript(position: Int) ->
AnyRegexOutput.Element { get }

    /// A type that represents a position
in the collection.
    ///
    /// Valid indices consist of the
position of every element and a
    /// "past the end" position that's
not valid for use as a subscript
    /// argument.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Index = Int

    /// A type that represents the
indices that are valid for subscripting
the
    /// collection, in ascending order.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Indices = Range<Int>

    /// A type that provides the
collection's iteration interface and
    /// encapsulates its iteration state.
    ///
    /// By default, a collection conforms
to the `Sequence` protocol by
    /// supplying `IndexingIterator` as
its associated `Iterator`
    /// type.

```

```

    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Iterator =
IndexingIterator<AnyRegexOutput>

    /// A collection representing a
contiguous subrange of this collection's
    /// elements. The subsequence shares
indices with the original collection.
    ///
    /// The default subsequence type for
collections that don't define their own
    /// is `Slice`.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias SubSequence =
Slice<AnyRegexOutput>
}

```

```

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension AnyRegexOutput {

```

```

    /// Accesses the capture with the
specified name, if a capture with that
name
    /// exists.
    ///
    /// - Parameter name: The name of the
capture to access.
    /// - Returns: An element providing
information about the capture, if there
is

```



```

        /// a capture named `name`;
    otherwise, `nil`.
    public subscript(name: String) ->
AnyRegexOutput.Element? { get }
}

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
public protocol
CustomConsumingRegexComponent :
RegexComponent {

```

```

    /// Process the input string within
the specified bounds, beginning at the
given index, and return
    /// the end position (upper bound) of
the match and the produced output.
    /// - Parameters:
    ///     - input: The string in which
the match is performed.
    ///     - index: An index of `input` at
which to begin matching.
    ///     - bounds: The bounds in `input`
in which the match is performed.
    /// - Returns: The upper bound where
the match terminates and a matched
instance, or `nil` if
    ///     there isn't a match.
    func consuming(_ input: String,
startingAt index: String.Index, in
bounds: Range<String.Index>) throws ->
(upperBound: String.Index, output:
Self.RegexOutput)?

```

```
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS  
9.0, tvOS 16.0, *)  
extension CustomConsumingRegexComponent {
```

```
    /// The regular expression  
    represented by this component.
```

```
    public var regex:  
    Regex<Self.RegexOutput> { get }  
}
```

```
/// A regular expression.
```

```
///
```

```
/// Regular expressions are a concise way  
of describing a pattern, which can  
/// help you match or extract portions of  
a string. You can create a `Regex`  
/// instance using regular expression  
syntax, either in a regex literal or a  
/// string.
```

```
///
```

```
///      // 'keyAndValue' is created using  
a regex literal
```

```
///      let keyAndValue = /(.+?): (.+)/
```

```
///      // 'simpleDigits' is created from  
a pattern in a string
```

```
///      let simpleDigits = try Regex("[0-  
9]+")
```

```
///
```

```
/// You can use a `Regex` to search for a  
pattern in a string or substring.
```

```
/// Call `contains(_:)` to check for the
```

```
presence of a pattern, or
/// `firstMatch(of:)` or `matches(of:)`
to find matches.
///
///     let setting = "color: 161 103
230"
///     if setting.contains(simpleDigits)
{
///         print("'\(setting)' contains
some digits.")
///     }
///     // Prints "'color: 161 103 230'
contains some digits."
///
/// When you find a match, the resulting
`Match` type includes an
/// `Match/output` property that
contains the matched substring along with
/// any captures:
///
///     if let match =
setting.firstMatch(of: keyAndValue) {
///         print("Key: \(match.1)")
///         print("Value: \(match.2)")
///     }
///     // Key: color
///     // Value: 161 103 230
///
/// When you import the `RegexBuilder`
module, you can also create `Regex`
/// instances using a clear and flexible
declarative syntax. Using this
/// style, you can combine, capture, and
```

```

transform regexes, `RegexBuilder`
/// types, and custom parsers.
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
public struct Regex<Output> :
RegexComponent {

    /// The regular expression
    represented by this component.
    public var regex: Regex<Output> { get
}

    /// The output type for this regular
    expression.
    ///
    /// A `Regex` instance's output type
    depends on whether the `Regex` has
    /// captures and how it is created.
    ///
    /// - A `Regex` created from a string
    using the ``init(_:)`` initializer
    /// has an output type of
    ``AnyRegexOutput``, whether it has
    captures or
    /// not.
    /// - A `Regex` without captures
    created from a regex literal, the
    /// ``init(_:as:)`` initializer, or
    a `RegexBuilder` closure has a
    /// `Substring` output type, where
    the substring is the portion of the
    /// string that was matched.
    /// - A `Regex` with captures created

```

from a regex literal or the
 /// ``init(_:as:)`` initializer has
a tuple of substrings as its output
 /// type. The first component of
the tuple is the full portion of the
string
 /// that was matched, with the
remaining components holding the
captures.

```
    @available(iOS 16.0, tvOS 16.0,  
watchOS 9.0, macOS 13.0, *)  
    public typealias RegexOutput = Output  
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS  
9.0, tvOS 16.0, *)  
extension Regex {
```

```
    /// The result of matching a regular  
expression against a string.
```

```
    ///  
    /// A `Match` forwards API to the  
`Output` generic parameter,  
    /// providing direct access to  
captures.
```

```
    @dynamicMemberLookup public struct  
Match {
```

```
        /// The range of the overall  
match.
```

```
        public let range:  
Range<String.Index>  
    }
```

```
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS  
9.0, tvOS 16.0, *)
```

```
extension Regex where Output ==  
AnyRegexOutput {
```

```
    /// Creates a regular expression from  
the given string, using a dynamic  
    /// capture list.
```

```
    ///  
    /// Use this initializer to create a  
`Regex` instance from a regular  
    /// expression that you have stored  
in `pattern`.
```

```
    ///  
    /// let simpleDigits = try  
Regex("[0-9]+")
```

```
    ///  
    /// This initializer throws an error  
if `pattern` uses invalid regular  
    /// expression syntax.
```

```
    ///  
    /// The output type of the new  
`Regex` is the dynamic  
`AnyRegexOutput`.
```

```
    /// If you know the capture structure  
of `pattern` ahead of time, use the  
    /// ``init(_:as:)`` initializer  
instead.
```

```
    ///  
    /// – Parameter pattern: A string  
with regular expression syntax.
```

```
    public init(_ pattern: String) throws
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex {
```

```
    /// Creates a regular expression from
the given string, using the specified
    /// capture type.
```

```
    ///
    /// You can use this initializer to
create a `Regex` instance from a regular
    /// expression that you have stored
in `pattern` when you know the capture
    /// structure of the regular
expression in advance.
```

```
    ///
    /// In this example, the regular
expression includes two parenthesized
    /// capture groups, so the capture
type is `(Substring, Substring,
Substring)`.
```

```
    /// The first substring in the tuple
represents the entire match, while the
    /// second and third substrings
represent the first and second capture
group,
```

```
    /// respectively.
    ///
    /// let keyAndValue = try
Regex("(.): (.+)", as: (Substring,
Substring, Substring).self)
```

```
    ///
    /// This initializer throws an error
    if `pattern` uses invalid regular
    /// expression syntax, or if
    `outputType` does not match the capture
    /// structure declared by `pattern`.
    If you don't know the capture structure
    /// in advance, use the ``init(_:)``
    initializer instead.
```

```
    ///
    /// - Parameters:
    ///   - pattern: A string with
    regular expression syntax.
    ///   - outputType: The desired type
    for the output captures.
```

```
    public init(_ pattern: String, as
    outputType: Output.Type = Output.self)
    throws
```

```
    /// Creates a regular expression that
    matches the given string exactly, as
    /// though every metacharacter in it
    was escaped.
```

```
    ///
    /// This example creates a regular
    expression that matches the string
    /// `(adj)`, including the
    parentheses. Although parentheses are
    regular
```

```
    /// expression metacharacters, they
    do not need escaping in the string passed
    /// as `verbatimString`.
    ///
```



```

    ///      let adjectiveDesignator =
Regex<Substring>(verbatim: "(adj.)")
    ///
    ///      print("awesome
(adj.)".contains(adjectiveDesignator))
    ///      // Prints "true"
    ///      print("apple
(n.)".contains(adjectiveDesignator))
    ///      // Prints "false"
    ///
    /// - Parameter verbatimString: A
string to convert into a regular
expression
    ///      exactly, escaping any
metacharacters.
    public init(verbatim verbatimString:
String)

    /// Returns a Boolean value
indicating whether a named capture with
the given
    /// name exists.
    ///
    /// This example shows a regular
expression that includes capture groups
    /// named `key` and `value`:
    ///
    ///      let regex = try
Regex("(?'key'.+?): (?'value'.+)" )
    ///      regex.contains(captureNamed:
"key")      // true
    ///      regex.contains(captureNamed:
"VALUE")    // false

```

```

        ///         regex.contains(captureNamed:
"1")           // false
        ///
        /// - Parameter name: The name to
look for among the regular expression's
        /// capture groups. Capture group
names are case sensitive.
        public func contains(captureNamed
name: String) -> Bool
    }

```

```

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex where Output ==
AnyRegexOutput {

```

```

    /// Creates a regular expression with
a dynamic capture list from the given
    /// regular expression.
    ///
    /// You can use this initializer to
convert a `Regex` with strongly-typed
    /// captures into a `Regex` with
`AnyRegexOutput` as its output type.
    ///
    /// - Parameter regex: A regular
expression to convert to use a dynamic
    /// capture list.
        public init<OtherOutput>(_ regex:
Regex<OtherOutput>)
    }

```

```

@available(macOS 13.0, iOS 16.0, watchOS

```

```
9.0, tvOS 16.0, *)  
extension Regex {
```

```
    /// Creates a regular expression with  
    a strongly-typed capture list from the  
    /// given regular expression.  
    ///  
    /// You can use this initializer to  
    convert a regular expression with a  
    /// dynamic capture list to one with  
    a strongly-typed capture list. If the  
    /// type you provide as `outputType`  
    doesn't match the capture structure of  
    /// `regex`, the initializer returns  
    `nil`.  
    ///  
    /// let dynamicRegex = try  
Regex("(.+?): (.+)")  
    /// if let stronglyTypedRegex =  
Regex(dynamicRegex, as: (Substring,  
Substring, Substring).self) {  
    /// print("Converted  
properly")  
    /// }  
    /// // Prints "Converted  
properly"  
    ///  
    /// - Parameters:  
    /// - regex: A regular expression  
to convert to use a strongly-typed  
capture  
    /// list.  
    /// - outputType: The capture
```

structure to use.

```
    public init?(_ regex:
Regex<AnyRegexOutput>, as outputType:
Output.Type = Output.self)
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex {
```

```
    /// Returns a match if this regex
matches the given string in its entirety.
```

```
    ///
    /// Call this method if you want the
regular expression to succeed only when
    /// it matches the entire string you
pass as `string`. The following example
    /// shows matching a regular
expression that only matches digits, with
    /// different candidate strings.
```

```
    ///
    ///      let digits = /[0-9]+/
    ///
    ///      if let digitsMatch = try
digits.wholeMatch(in: "2022") {
    ///          print(digitsMatch.0)
    ///      } else {
    ///          print("No match.")
    ///      }
    ///      // Prints "2022"
```

```
    ///
    ///      if let digitsMatch = try
digits.wholeMatch(in: "The year is
```

```

2022.") {
    ///          print(digitsMatch.0)
    ///      } else {
    ///          print("No match.")
    ///      }
    ///      // Prints "No match."
    ///
    /// The `wholeMatch(in:)` method can
    throw an error if this regex includes
    /// a transformation closure that
    throws an error.
    ///
    /// - Parameter string: The string to
    match this regular expression against.
    /// - Returns: The match, if this
    regex matches the entirety of `string`;
    /// otherwise, `nil`.
    public func wholeMatch(in string:
String) throws -> Regex<Output>.Match?

    /// Returns a match if this regex
    matches the given string at its start.
    ///
    /// Call this method if you want the
    regular expression to succeed only when
    /// it matches only at the start of
    the given string. This example uses
    /// `prefixMatch(in:)` and a regex
    that matches a title-case word to search
    /// for such a word at the start of
    different strings:
    ///
    ///      let titleCaseWord = /[A-Z][A-

```

```

Za-z]+/
    ///
    ///         if let wordMatch = try
titleCaseWord.prefixMatch(in: "Searching
in a Regex") {
    ///             print(wordMatch.0)
    ///         } else {
    ///             print("No match.")
    ///         }
    ///         // Prints "Searching"
    ///
    ///         if let wordMatch = try
titleCaseWord.prefixMatch(in: "title case
word at the End") {
    ///             print(wordMatch.0)
    ///         } else {
    ///             print("No match.")
    ///         }
    ///         // Prints "No match."
    ///
    /// The `prefixMatch(in:)` method can
throw an error if this regex includes
    /// a transformation closure that
throws an error.
    ///
    /// - Parameter string: The string to
match this regular expression against.
    /// - Returns: The match, if this
regex matches at the start of `string`;
    /// otherwise, `nil`.
    public func prefixMatch(in string:
String) throws -> Regex<Output>.Match?

```

```

    /// Returns the first match for this
    regex found in the given string.
    ///
    /// Use the `firstMatch(in:)` method
    to search for the first occurrence of
    /// this regular expression in
    `string`. This example searches for the
    first
    /// sequence of digits that occurs in
    a string:
    ///
    ///         let digits = /[0-9]+/
    ///
    ///         if let digitsMatch = try
    digits.firstMatch(in: "The year is 2022;
    last year was 2021.") {
    ///             print(digitsMatch.0)
    ///         } else {
    ///             print("No match.")
    ///         }
    ///         // Prints "2022"
    ///
    /// The `firstMatch(in:)` method can
    throw an error if this regex includes
    /// a transformation closure that
    throws an error.
    ///
    /// - Parameter string: The string to
    match this regular expression against.
    /// - Returns: The match, if one is
    found; otherwise, `nil`.
    public func firstMatch(in string:
    String) throws -> Regex<Output>.Match?

```

```
    /// Returns a match if this regex
matches the given substring in its
entirety.
    ///
    /// Call this method if you want the
regular expression to succeed only when
    /// it matches the entire string you
pass as `string`. The following example
    /// shows matching a regular
expression that only matches digits, with
    /// different candidate strings.
    ///
    ///      let digits = /[0-9]+/
    ///
    ///      if let digitsMatch = try
digits.wholeMatch(in: "2022") {
    ///          print(digitsMatch.0)
    ///      } else {
    ///          print("No match.")
    ///      }
    ///      // Prints "2022"
    ///
    ///      if let digitsMatch = try
digits.wholeMatch(in: "The year is
2022.") {
    ///          print(digitsMatch.0)
    ///      } else {
    ///          print("No match.")
    ///      }
    ///      // Prints "No match."
    ///
    /// The `wholeMatch(in:)` method can
```



```

throw an error if this regex includes
    /// a transformation closure that
throws an error.
    ///
    /// - Parameter string: The substring
to match this regular expression
    /// against.
    /// - Returns: The match, if this
regex matches the entirety of `string`;
    /// otherwise, `nil`.
    public func wholeMatch(in string:
Substring) throws -> Regex<Output>.Match?

    /// Returns a match if this regex
matches the given substring at its start.
    ///
    /// Call this method if you want the
regular expression to succeed only when
    /// it matches only at the start of
the given string. This example uses
    /// `prefixMatch(in:)` and a regex
that matches a title-case word to search
    /// for such a word at the start of
different strings:
    ///
    /// let titleCaseWord = /[A-Z][A-
Za-z]+/
    ///
    /// if let wordMatch = try
titleCaseWord.prefixMatch(in: "Searching
in a Regex") {
    /// print(wordMatch.0)
    /// } else {

```

```

        ///          print("No match.")
        ///      }
        ///      // Prints "Searching"
        ///
        ///      if let wordMatch = try
titleCaseWord.prefixMatch(in: "title case
word at the End") {
        ///          print(wordMatch.0)
        ///      } else {
        ///          print("No match.")
        ///      }
        ///      // Prints "No match."
        ///
        /// The `prefixMatch(in:)` method can
throw an error if this regex includes
        /// a transformation closure that
throws an error.
        ///
        /// - Parameter string: The substring
to match this regular expression
        /// against.
        /// - Returns: The match, if this
regex matches at the start of `string`;
        /// otherwise, `nil`.
        public func prefixMatch(in string:
Substring) throws -> Regex<Output>.Match?

        /// Returns the first match for this
regex found in the given substring.
        ///
        /// Use the `firstMatch(in:)` method
to search for the first occurrence of
        /// this regular expression in

```

`string`. This example searches for the first

sequence of digits that occurs in a string:

```
///
///      let digits = /[0-9]+/
///
///      if let digitsMatch = try
digits.firstMatch(in: "The year is 2022;
last year was 2021.") {
///          print(digitsMatch.0)
///      } else {
///          print("No match.")
///      }
///      // Prints "2022"
///
/// The `firstMatch(in:)` method can
throw an error if this regex includes
/// a transformation closure that
throws an error.
///
/// - Parameter string: The substring
to match this regular expression
/// against.
/// - Returns: The match, if one is
found; otherwise, `nil`.
    public func firstMatch(in string:
Substring) throws -> Regex<Output>.Match?
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex {
```

```
    /// Returns a regular expression that  
    ignores case when matching.
```

```
    ///  
    /// - Parameter ignoresCase: A  
    Boolean value indicating whether to  
    ignore case.
```

```
    /// - Returns: The modified regular  
    expression.
```

```
    public func ignoresCase(_  
    ignoresCase: Bool = true) ->  
    Regex<Regex<Output>.RegexOutput>
```

```
    /// Returns a regular expression that  
    matches only ASCII characters as word
```

```
    /// characters.
```

```
    ///
```

```
    /// - Parameter useASCII: A Boolean  
    value indicating whether to match only
```

```
    /// ASCII characters as word  
    characters.
```

```
    /// - Returns: The modified regular  
    expression.
```

```
    public func asciiOnlyWordCharacters(_  
    useASCII: Bool = true) ->  
    Regex<Regex<Output>.RegexOutput>
```

```
    /// Returns a regular expression that  
    matches only ASCII characters as digits.
```

```
    ///
```

```
    /// - Parameter useasciiOnlyDigits: A  
    Boolean value indicating whether to
```

```
    /// match only ASCII characters as
```

```

digits.
    /// - Returns: The modified regular
expression.
    public func asciiOnlyDigits(_
useASCII: Bool = true) ->
Regex<Regex<Output>.RegexOutput>

    /// Returns a regular expression that
matches only ASCII characters as space
    /// characters.
    ///
    /// - Parameter asciiOnlyWhitespace:
A Boolean value indicating whether to
    /// match only ASCII characters as
space characters.
    /// - Returns: The modified regular
expression.
    public func asciiOnlyWhitespace(_
useASCII: Bool = true) ->
Regex<Regex<Output>.RegexOutput>

    /// Returns a regular expression that
matches only ASCII characters when
    /// matching character classes.
    ///
    /// - Parameter useASCII: A Boolean
value indicating whether to match only
    /// ASCII characters when matching
character classes.
    /// - Returns: The modified regular
expression.
    public func
asciiOnlyCharacterClasses(_ useASCII:

```

```
Bool = true) ->  
Regex<Regex<Output>.RegexOutput>
```

```
    /// Returns a regular expression that  
    uses the specified word boundary  
    algorithm.
```

```
    ///  
    /// - Parameter wordBoundaryKind: The  
    algorithm to use for determining word  
    boundaries.
```

```
    /// - Returns: The modified regular  
    expression.
```

```
    public func wordBoundaryKind(_  
wordBoundaryKind: RegexWordBoundaryKind)  
-> Regex<Regex<Output>.RegexOutput>
```

```
    /// Returns a regular expression  
    where the "any" metacharacter (`.`)
```

```
    /// also matches against the start  
    and end of a line.
```

```
    ///  
    /// - Parameter dotMatchesNewlines: A  
    Boolean value indicating whether `.`
```

```
    /// should match a newline  
    character.
```

```
    /// - Returns: The modified regular  
    expression.
```

```
    public func dotMatchesNewlines(_  
dotMatchesNewlines: Bool = true) ->  
Regex<Regex<Output>.RegexOutput>
```

```
    /// Returns a regular expression  
    where the start and end of input
```

```
    /// anchors (`^` and `$`) also match  
    against the start and end of a line.
```

```
    ///  
    /// This method corresponds to  
    applying the `m` option in regex syntax.  
    For
```

```
    /// this behavior in the  
    `RegexBuilder` syntax, see  
    /// `Anchor.startOfLine`,  
    `Anchor.endOfLine`,  
    `Anchor.startOfSubject`,  
    /// and `Anchor.endOfSubject`.
```

```
    ///  
    /// - Parameter matchLineEndings: A  
    Boolean value indicating whether `^` and  
    /// `$` should match the start and  
    end of lines, respectively.
```

```
    /// - Returns: The modified regular  
    expression.
```

```
    public func anchorsMatchLineEndings(_  
matchLineEndings: Bool = true) ->  
Regex<Regex<Output>.RegexOutput>
```

```
    /// Returns a regular expression  
    where quantifiers use the specified  
    behavior
```

```
    /// by default.  
    ///  
    /// This setting does not affect  
    calls to quantifier methods, such as  
    /// `OneOrMore`, that include an  
    explicit `behavior` parameter.
```

```
    ///
```

```
    /// Passing `.eager` or `.reluctant`
to this method corresponds to applying
    /// the ?(?-U) or ?(?U) option in
regex syntax, respectively.
    ///
    /// - Parameter behavior: The default
behavior to use for quantifiers.
    public func repetitionBehavior(_
behavior: RegexRepetitionBehavior) ->
Regex<Regex<Output>.RegexOutput>

    /// Returns a regular expression that
matches with the specified semantic
    /// level.
    ///
    /// When matching with grapheme
cluster semantics (the default),
    /// metacharacters like `.` and `\w`,
custom character classes, and character
    /// class instances like `.any` match
a grapheme cluster when possible,
    /// corresponding with the default
string representation. In addition,
    /// matching with grapheme cluster
semantics compares characters using their
    /// canonical representation,
corresponding with how strings comparison
works.
    ///
    /// When matching with Unicode scalar
semantics, metacharacters and character
    /// classes always match a single
Unicode scalar value, even if that scalar
```



```

    /// comprises part of a grapheme
cluster.
    ///
    /// These semantic levels can lead to
different results, especially when
    /// working with strings that have
decomposed characters. In the following
    /// example, `queRegex` matches any
3-character string that begins with
`"q"`.
    ///
    ///     let composed = "qué"
    ///     let decomposed = "que\u{301}"
    ///
    ///     let queRegex = /^q..$/
    ///
    ///
print(composed.contains(queRegex))
    ///     // Prints "true"
    ///
print(decomposed.contains(queRegex))
    ///     // Prints "true"
    ///
    /// When using Unicode scalar
semantics, however, the regular
expression only
    /// matches the composed version of
the string, because each `.` matches a
    /// single Unicode scalar value.
    ///
    ///     let queRegexScalar =
queRegex.matchingSemantics(.unicodeScalar
)

```

```

    ///
print(composed.contains(queRegexScalar))
    ///      // Prints "true"
    ///
print(decomposed.contains(queRegexScalar)
)
    ///      // Prints "false"
    ///
    /// - Parameter semanticLevel: The
semantics to use during matching.
    /// - Returns: The modified regular
expression.
    public func matchingSemantics(_
semanticLevel: RegexSemanticLevel) ->
Regex<Regex<Output>.RegexOutput>
}

```

```

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex.Match where Output ==
AnyRegexOutput {

```

```

    /// Accesses the capture with the
specified name, if a capture with that
name
    /// exists.
    ///
    /// - Parameter name: The name of the
capture to access.
    /// - Returns: An element providing
information about the capture, if there
is
    /// a capture named `name`;

```

```

otherwise, `nil`.
    public subscript(name: String) ->
AnyRegexOutput.Element? { get }
}

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex.Match where Output ==
AnyRegexOutput {

    /// Creates a regular expression
match with a dynamic capture list from
the
    /// given match.
    ///
    /// You can use this initializer to
convert a `Regex.Match` with
    /// strongly-typed captures into a
match with the type-eraser
`AnyRegexOutput`
    /// as its output type.
    ///
    /// - Parameter match: A regular
expression match to convert to a match
with
    /// type-erased captures.
    public init<OtherOutput>(_ match:
Regex<OtherOutput>.Match)
}

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Regex.Match {

```

```

    /// The output produced from the
    match operation.
    public var output: Output { get }

    /// Accesses a capture by its name or
    number.
    public subscript<T>(dynamicMember
    keyPath: KeyPath<Output, T>) -> T { get }
}

/// A type that represents a regular
    expression.
    ///
    /// You can use types that conform to
    `RegexComponent` as parameters to string
    /// searching operations and inside
    `RegexBuilder` closures.
    @available(macOS 13.0, iOS 16.0, watchOS
    9.0, tvOS 16.0, *)
    public protocol
    RegexComponent<RegexOutput> {

        /// The output type for this regular
        expression.
        ///
        /// A `Regex` instance's output type
        depends on whether the `Regex` has
        /// captures and how it is created.
        ///
        /// - A `Regex` created from a string
        using the ``init(_:)`` initializer
        /// has an output type of

```

```
`AnyRegexOutput`, whether it has captures or
    /// not.
    /// - A `Regex` without captures created from a regex literal, the
    /// ``init(_:as:)`` initializer, or a `RegexBuilder` closure has a
    /// `Substring` output type, where the substring is the portion of the
    /// string that was matched.
    /// - A `Regex` with captures created from a regex literal or the
    /// ``init(_:as:)`` initializer has a tuple of substrings as its output
    /// type. The first component of the tuple is the full portion of the
    /// string
    /// that was matched, with the remaining components holding the
    /// captures.
```

associatedtype **RegexOutput**

```
    /// The regular expression represented by this component.
    var regex: Regex<Self.RegexOutput> {
get }
}
```

```
/// Specifies how much to attempt to match when using a quantifier.
```

```
///
```

```
/// See `Regex/repetitionBehavior(_)` for more about specifying the default
```

```

/// matching behavior for all or part of
a regex.
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
public struct RegexRepetitionBehavior :
Hashable {

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    /// don't call `finalize()` on the
`hasher` instance provided,
    /// or replace it with a different
instance.
    /// Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    /// of this instance.
    public func hash(into hasher: inout
Hasher)

```

```
    /// Returns a Boolean value
    indicating whether two values are equal.
    ///
    /// Equality is the inverse of
    inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
    `false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
    compare.
```

```
    public static func == (a:
    RegexRepetitionBehavior, b:
    RegexRepetitionBehavior) -> Bool
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
    be equal across different executions of
    /// your program. Do not save hash
    values to use during a future execution.
```

```
    ///
    /// - Important: `hashValue` is
    deprecated as a `Hashable` requirement.
    To
```

```
    /// conform to `Hashable`,
    implement the `hash(into:)` requirement
    instead.
```

```
    /// The compiler provides an
    implementation for `hashValue` for you.
```

```
    public var hashValue: Int { get }
```

```
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS  
9.0, tvOS 16.0, *)
```

```
extension RegexRepetitionBehavior {
```

```
    /// Match as much of the input string  
    as possible, backtracking when  
    /// necessary.
```

```
    public static var eager:  
RegexRepetitionBehavior { get }
```

```
    /// Match as little of the input  
    string as possible, expanding the matched  
    /// region as necessary to complete a  
    match.
```

```
    public static var reluctant:  
RegexRepetitionBehavior { get }
```

```
    /// Match as much of the input string  
    as possible, performing no backtracking.
```

```
    public static var possessive:  
RegexRepetitionBehavior { get }  
}
```

```
/// A semantic level to use during regex  
matching.
```

```
///
```

```
/// The semantic level determines whether  
a regex matches with the same
```

```
/// character-based semantics as string  
comparisons or by matching individual
```

```
/// Unicode scalar values. See
```


`Regex/matchingSemantics(_:)` for more about

/// changing the semantic level for all or part of a regex.

@available(macOS 13.0, iOS 16.0, watchOS 9.0, tvOS 16.0, *)

public struct RegexSemanticLevel : Hashable {

/// Match at the character level.

///

/// At this semantic level, each matched element is a `Character` value.

/// This is the default semantic level.

public static var graphemeCluster: RegexSemanticLevel { get }

/// Match at the Unicode scalar level.

///

/// At this semantic level, the string's `UnicodeScalarView` is used for

/// matching, and each matched element is a `UnicodeScalar` value.

public static var unicodeScalar: RegexSemanticLevel { get }

/// Hashes the essential components of this value by feeding them into the
 /// given hasher.

///

/// Implement this method to conform

```

to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    /// don't call `finalize()` on the
`hasher` instance provided,
    /// or replace it with a different
instance.
    /// Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    /// of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///     - lhs: A value to compare.
    ///     - rhs: Another value to

```

compare.

```
    public static func == (a:
RegexSemanticLevel, b:
RegexSemanticLevel) -> Bool
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    /// conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    /// The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}
```

```
/// A word boundary algorithm to use
during regex matching.
///
/// See ``Regex/wordBoundaryKind(_:)``
for information about specifying the
/// word boundary kind for all or part of
a regex.
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
public struct RegexWordBoundaryKind :
```

Hashable {

```
    /// A word boundary algorithm that
    implements the "simple word boundary"
    /// Unicode recommendation.
    ///
    /// A simple word boundary is a
    position in the input between two
    characters
    /// that match `/\w\W/` or `/\W\w/`,
    or between the start or end of the input
    /// and a `\w` character. Word
    boundaries therefore depend on the
    option-
```

```
    /// defined behavior of `\w`.
    public static var simple:
RegexWordBoundaryKind { get }
```

```
    /// A word boundary algorithm that
    implements the "default word boundary"
    /// Unicode recommendation.
    ///
    /// Default word boundaries use a
    Unicode algorithm that handles some cases
    /// better than simple word
    boundaries, such as words with internal
    /// punctuation, changes in script,
    and Emoji.
```

```
    public static var `default`:
RegexWordBoundaryKind { get }
```

```
    /// Hashes the essential components
    of this value by feeding them into the
```

```

    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    /// don't call `finalize()` on the
`hasher` instance provided,
    /// or replace it with a different
instance.
    /// Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    /// of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///

```

```

    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (a:
RegexWordBoundaryKind, b:
RegexWordBoundaryKind) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    /// conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    /// The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

```

```

extension Collection where Self.Element :
Equatable {

```

```

    /// Returns a Boolean value
indicating whether the collection
contains the
    /// given sequence.

```

```
    /// - Parameter other: A sequence to
search for within this collection.
```

```
    /// - Returns: `true` if the
collection contains the specified
sequence,
```

```
    /// otherwise `false`.
```

```
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
```

```
    public func contains<C>(_ other: C)
-> Bool where C : Collection,
Self.Element == C.Element
}
```

```
extension StringProtocol {
```

```
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
```

```
    public func contains(_ other: String)
-> Bool
```

```
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
```

```
    public func contains(_ other:
Substring) -> Bool
}
```

```
extension BidirectionalCollection where
Self.SubSequence == Substring {
```

```
    /// Returns a Boolean value
indicating whether the collection
contains the
```

```
    /// given regex.
```

```

    /// - Parameter regex: A regex to
    search for within this collection.
    /// - Returns: `true` if the regex
    was found in the collection, otherwise
    /// `false`.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func contains(_ regex: some
    RegexComponent) -> Bool
}

```

```

extension Collection where Self.Element :
Equatable {

```

```

    /// Finds and returns the range of
    the first occurrence of a given
    collection
    /// within this collection.
    ///
    /// - Parameter other: The collection
    to search for.
    /// - Returns: A range in the
    collection of the first occurrence of
    `sequence`.
    /// Returns nil if `sequence` is not
    found.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func firstRange<C>(of other:
    C) -> Range<Self.Index>? where C :
    Collection, Self.Element == C.Element
}

```



```

extension BidirectionalCollection where
Self.Element : Comparable {

    /// Finds and returns the range of
    the first occurrence of a given
    collection
    /// within this collection.
    ///
    /// - Parameter other: The collection
    to search for.
    /// - Returns: A range in the
    collection of the first occurrence of
    `sequence`.
    /// Returns `nil` if `sequence` is
    not found.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func firstRange<C>(of other:
    C) -> Range<Self.Index>? where C :
    Collection, Self.Element == C.Element
}

```

```

extension BidirectionalCollection where
Self.SubSequence == Substring {

```

```

    /// Finds and returns the range of
    the first occurrence of a given regex
    /// within the collection.
    /// - Parameter regex: The regex to
    search for.
    /// - Returns: A range in the
    collection of the first occurrence of
    `regex`.

```

```

    /// Returns `nil` if `regex` is not
    found.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func firstRange(of regex: some
    RegexComponent) -> Range<Self.Index>?
}

```

```

extension Collection where Self.Element :
    Equatable {

```

```

    /// Finds and returns the ranges of
    the all occurrences of a given sequence
    /// within the collection.
    /// - Parameter other: The sequence
    to search for.
    /// - Returns: A collection of ranges
    of all occurrences of `other`. Returns
    /// an empty collection if `other`
    is not found.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func ranges<C>(of other: C) ->
    [Range<Self.Index>] where C : Collection,
    Self.Element == C.Element
}

```

```

extension BidirectionalCollection where
    Self.SubSequence == Substring {

```

```

    /// Finds and returns the ranges of
    the all occurrences of a given sequence
    /// within the collection.

```

```

    ///
    /// - Parameter regex: The regex to
search for.
    /// - Returns: A collection or ranges
in the receiver of all occurrences of
    /// `regex`. Returns an empty
collection if `regex` is not found.
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public func ranges(of regex: some
RegexComponent) -> [Range<Self.Index>]
}

```

```

extension RangeReplaceableCollection
where Self.Element : Equatable {

```

```

    /// Returns a new collection in which
all occurrences of a target sequence
    /// are replaced by another
collection.
    /// - Parameters:
    ///     - other: The sequence to
replace.
    ///     - replacement: The new elements
to add to the collection.
    ///     - subrange: The range in the
collection in which to search for
`other`.
    ///     - maxReplacements: A number
specifying how many occurrences of
`other`
    ///     to replace. Default is
`Int.max`.

```

```

    /// - Returns: A new collection in
    which all occurrences of `other` in
    /// `subrange` of the collection are
    replaced by `replacement`.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func replacing<C,
    Replacement>(_ other: C, with
    replacement: Replacement, subrange:
    Range<Self.Index>, maxReplacements: Int =
    .max) -> Self where C : Collection,
    Replacement : Collection, Self.Element ==
    C.Element, C.Element ==
    Replacement.Element

```

```

    /// Returns a new collection in which
    all occurrences of a target sequence
    /// are replaced by another
    collection.

```

```

    /// - Parameters:
    ///     - other: The sequence to
    replace.

```

```

    ///     - replacement: The new elements
    to add to the collection.

```

```

    ///     - maxReplacements: A number
    specifying how many occurrences of
    `other`

```

```

    ///     to replace. Default is
    `Int.max`.

```

```

    /// - Returns: A new collection in
    which all occurrences of `other` in
    /// `subrange` of the collection are
    replaced by `replacement`.

```

```

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public func replacing<C,
Replacement>(_ other: C, with
replacement: Replacement,
maxReplacements: Int = .max) -> Self
where C : Collection, Replacement :
Collection, Self.Element == C.Element,
C.Element == Replacement.Element

```

```

    /// Replaces all occurrences of a
target sequence with a given collection
    /// - Parameters:
    ///     - other: The sequence to
replace.
    ///     - replacement: The new elements
to add to the collection.
    ///     - maxReplacements: A number
specifying how many occurrences of
`other`
    ///     to replace. Default is
`Int.max`.

```

```

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public mutating func replace<C,
Replacement>(_ other: C, with
replacement: Replacement,
maxReplacements: Int = .max) where C :
Collection, Replacement : Collection,
Self.Element == C.Element, C.Element ==
Replacement.Element
}

```

```

extension RangeReplaceableCollection
where Self.SubSequence == Substring {

    /// Returns a new collection in which
    all occurrences of a sequence matching
    /// the given regex are replaced by
    another collection.
    /// - Parameters:
    ///     - regex: A regex describing the
    sequence to replace.
    ///     - replacement: The new elements
    to add to the collection.
    ///     - subrange: The range in the
    collection in which to search for
    `regex`.
    ///     - maxReplacements: A number
    specifying how many occurrences of the
    /// sequence matching `regex` to
    replace. Default is `Int.max`.
    /// - Returns: A new collection in
    which all occurrences of subsequence
    /// matching `regex` in `subrange`
    are replaced by `replacement`.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func replacing<Replacement>(_
    regex: some RegexComponent, with
    replacement: Replacement, subrange:
    Range<Self.Index>, maxReplacements: Int =
    .max) -> Self where Replacement :
    Collection, Replacement.Element ==
    Character

```

```
    /// Returns a new collection in which  
all occurrences of a sequence matching  
    /// the given regex are replaced by  
another collection.
```

```
    /// - Parameters:  
    ///     - regex: A regex describing the  
sequence to replace.
```

```
    ///     - replacement: The new elements  
to add to the collection.
```

```
    ///     - maxReplacements: A number  
specifying how many occurrences of the  
    ///     sequence matching `regex` to  
replace. Default is `Int.max`.
```

```
    /// - Returns: A new collection in  
which all occurrences of subsequence  
    /// matching `regex` are replaced by  
`replacement`.
```

```
    @available(macOS 13.0, iOS 16.0,  
watchOS 9.0, tvOS 16.0, *)  
    public func replacing<Replacement>(_  
regex: some RegexComponent, with  
replacement: Replacement,  
maxReplacements: Int = .max) -> Self  
where Replacement : Collection,  
Replacement.Element == Character
```

```
    /// Replaces all occurrences of the  
sequence matching the given regex with  
    /// a given collection.
```

```
    /// - Parameters:  
    ///     - regex: A regex describing the  
sequence to replace.
```

```
    ///     - replacement: The new elements
```

to add to the collection.

```
    /// - maxReplacements: A number
specifying how many occurrences of the
    /// sequence matching `regex` to
replace. Default is `Int.max`.
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public mutating func
replace<Replacement>(_ regex: some
RegexComponent, with replacement:
Replacement, maxReplacements: Int = .max)
where Replacement : Collection,
Replacement.Element == Character
}
```

```
extension Collection where Self.Element :
Equatable {
```

```
    /// Returns the longest possible
subsequences of the collection, in order,
    /// around elements equal to the
given separator.
```

```
    ///
    /// - Parameter separator: The
element to be split upon.
```

```
    /// - Returns: A collection of
subsequences, split from this
collection's
```

```
    /// elements.
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public func split<C>(separator: C,
maxSplits: Int = .max,
```



```
omittingEmptySubsequences: Bool = true)
-> [Self.SubSequence] where C :
Collection, Self.Element == C.Element
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension BidirectionalCollection where
Self.SubSequence == Substring {
```

```
    /// Returns the longest possible
subsequences of the collection, in order,
    /// around elements equal to the
given separator.
```

```
    ///
    /// - Parameter separator: A regex
describing elements to be split upon.
    /// - Returns: A collection of
substrings, split from this collection's
    /// elements.
```

```
    public func split(separator: some
RegexComponent, maxSplits: Int = .max,
omittingEmptySubsequences: Bool = true)
-> [Self.SubSequence]
}
```

```
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension BidirectionalCollection where
Self.SubSequence == Substring {
```

```
    /// Returns a Boolean value
indicating whether the initial elements
```

of the

```
    /// sequence are the same as the  
    elements in the specified regex.
```

```
    ///
```

```
    /// - Parameter regex: A regex to  
    compare to this sequence.
```

```
    /// - Returns: `true` if the initial  
    elements of the sequence matches the
```

```
    /// beginning of `regex`;  
    otherwise, `false`.
```

```
    public func starts(with regex: some  
    RegexComponent) -> Bool  
}
```

```
extension Collection {
```

```
    @available(macOS 13.0, iOS 16.0,  
    watchOS 9.0, tvOS 16.0, *)
```

```
    public func trimmingPrefix(while  
    predicate: (Self.Element) throws -> Bool)  
    rethrows -> Self.SubSequence  
}
```

```
extension Collection where Self ==  
Self.SubSequence {
```

```
    @available(macOS 13.0, iOS 16.0,  
    watchOS 9.0, tvOS 16.0, *)
```

```
    public mutating func trimPrefix(while  
    predicate: (Self.Element) throws -> Bool)  
    throws  
}
```

```

extension RangeReplaceableCollection {

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public mutating func trimPrefix(while
predicate: (Self.Element) throws -> Bool)
rethrows
}

```

```

extension Collection where Self.Element :
Equatable {

```

```

    /// Returns a new collection of the
    same type by removing `prefix` from the
    start
    /// of the collection.
    /// - Parameter prefix: The
    collection to remove from this
    collection.
    /// - Returns: A collection
    containing the elements of the collection
    that are
    /// not removed by `prefix`.
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public func trimmingPrefix<Prefix>(_
prefix: Prefix) -> Self.SubSequence where
Prefix : Sequence, Self.Element ==
Prefix.Element
}

```

```

extension Collection where Self ==
Self.SubSequence, Self.Element :

```

```
Equatable {
```

```
    /// Removes `prefix` from the start  
    of the collection.
```

```
    /// - Parameter prefix: The  
    collection to remove from this  
    collection.
```

```
    @available(macOS 13.0, iOS 16.0,  
watchOS 9.0, tvOS 16.0, *)  
    public mutating func  
trimPrefix<Prefix>(_ prefix: Prefix)  
where Prefix : Sequence, Self.Element ==  
Prefix.Element  
}
```

```
extension RangeReplaceableCollection  
where Self.Element : Equatable {
```

```
    /// Removes `prefix` from the start  
    of the collection.
```

```
    /// - Parameter prefix: The  
    collection to remove from this  
    collection.
```

```
    @available(macOS 13.0, iOS 16.0,  
watchOS 9.0, tvOS 16.0, *)  
    public mutating func  
trimPrefix<Prefix>(_ prefix: Prefix)  
where Prefix : Sequence, Self.Element ==  
Prefix.Element  
}
```

```
extension BidirectionalCollection where  
Self.SubSequence == Substring {
```

```

    /// Returns a new collection of the
    same type by removing the initial
    elements
    /// that matches the given regex.
    /// - Parameter regex: The regex to
    remove from this collection.
    /// - Returns: A collection
    containing the elements that does not
    match
    /// `regex` from the start.
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public func trimmingPrefix(_ regex:
some RegexComponent) -> Self.SubSequence
}

```

```

extension RangeReplaceableCollection
where Self : BidirectionalCollection,
Self.SubSequence == Substring {

```

```

    /// Removes the initial elements that
    matches the given regex.
    /// - Parameter regex: The regex to
    remove from this collection.
    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public mutating func trimPrefix(_
regex: some RegexComponent)
}

```

```

extension BidirectionalCollection where
Self.SubSequence == Substring {

```

```

    /// Returns the first match of the
    specified regex within the collection.
    /// - Parameter regex: The regex to
    search for.
    /// - Returns: The first match of
    `regex` in the collection, or `nil` if
    /// there isn't a match.
    @available(macOS 13.0, iOS 16.0,
    watchOS 9.0, tvOS 16.0, *)
    public func firstMatch<Output>(of r:
    some RegexComponent) ->
    Regex<Output>.Match?
}

```

```

extension RangeReplaceableCollection
where Self.SubSequence == Substring {

```

```

    /// Returns a new collection in which
    all occurrences of a sequence matching
    /// the given regex are replaced by
    another regex match.
    /// - Parameters:
    ///     - regex: A regex describing the
    sequence to replace.
    ///     - subrange: The range in the
    collection in which to search for
    `regex`.
    ///     - maxReplacements: A number
    specifying how many occurrences of the
    ///     sequence matching `regex` to
    replace. Default is `Int.max`.
    ///     - replacement: A closure that

```

```
receives the full match information,  
    /// including captures, and returns  
a replacement collection.
```

```
    /// - Returns: A new collection in  
which all occurrences of subsequence  
    /// matching `regex` are replaced by  
`replacement`.
```

```
    @available(macOS 13.0, iOS 16.0,  
watchOS 9.0, tvOS 16.0, *)  
    public func replacing<Output,  
Replacement>(_ regex: some  
RegexComponent, subrange:  
Range<Self.Index>, maxReplacements: Int =  
.max, with replacement:  
(Regex<Output>.Match) throws ->  
Replacement) rethrows -> Self where  
Replacement : Collection,  
Replacement.Element == Character
```

```
    /// Returns a new collection in which  
all occurrences of a sequence matching  
    /// the given regex are replaced by  
another collection.
```

```
    /// - Parameters:  
    /// - regex: A regex describing the  
sequence to replace.
```

```
    /// - maxReplacements: A number  
specifying how many occurrences of the  
    /// sequence matching `regex` to  
replace. Default is `Int.max`.
```

```
    /// - replacement: A closure that  
receives the full match information,  
    /// including captures, and returns
```

a replacement collection.

/// - Returns: A new collection in
which all occurrences of subsequence
/// matching `regex` are replaced by
`replacement`.

```
@available(macOS 13.0, iOS 16.0,  
watchOS 9.0, tvOS 16.0, *)  
public func replacing<Output,  
Replacement>(_ regex: some  
RegexComponent, maxReplacements: Int  
= .max, with replacement:  
(Regex<Output>.Match) throws ->  
Replacement) rethrows -> Self where  
Replacement : Collection,  
Replacement.Element == Character
```

/// Replaces all occurrences of the
sequence matching the given regex with
/// a given collection.
/// - Parameters:
/// - regex: A regex describing the
sequence to replace.

/// - maxReplacements: A number
specifying how many occurrences of the
/// sequence matching `regex` to
replace. Default is `Int.max`.

/// - replacement: A closure that
receives the full match information,
/// including captures, and returns
a replacement collection.

```
@available(macOS 13.0, iOS 16.0,  
watchOS 9.0, tvOS 16.0, *)  
public mutating func replace<Output,
```



```

Replacement>(_ regex: some
RegexComponent, maxReplacements: Int
= .max, with replacement:
(Regex<Output>.Match) throws ->
Replacement) rethrows where Replacement :
Collection, Replacement.Element ==
Character
}

```

```

extension BidirectionalCollection where
Self.SubSequence == Substring {

```

```

    /// Returns a collection containing
    all matches of the specified regex.

```

```

    /// - Parameter regex: The regex to
    search for.

```

```

    /// - Returns: A collection of
    matches of `regex`.

```

```

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)

```

```

    public func matches<Output>(of r:
some RegexComponent) ->
[Regex<Output>.Match]
}

```

```

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)

```

```

extension BidirectionalCollection where
Self.SubSequence == Substring {

```

```

    /// Returns a match if this string is
    matched by the given regex in its
    entirety.

```

```

    ///
    /// - Parameter regex: The regular
expression to match.
    /// - Returns: The match, if one is
found. If there is no match, or a
    /// transformation in `regex`
throws an error, this method returns
`nil`.
    public func wholeMatch<R>(of regex:
R) -> Regex<R.RegexOutput>.Match? where R
: RegexComponent

    /// Returns a match if this string is
matched by the given regex at its start.
    ///
    /// - Parameter regex: The regular
expression to match.
    /// - Returns: The match, if one is
found. If there is no match, or a
    /// transformation in `regex`
throws an error, this method returns
`nil`.
    public func prefixMatch<R>(of regex:
R) -> Regex<R.RegexOutput>.Match? where R
: RegexComponent
}

```