String Methods

Sr.No	Methods with Description
1	**char charAt(int index)** Returns the character at the specified index.
2	**int compareTo(Object o)** Compares this String to another Object.
3	**int compareTo(String anotherString)** Compares two strings lexicographically.
4	**int compareToIgnoreCase(String str)z** Compares two strings lexicographically, ignoring case differences.
5	**String concat(String str)** Concatenates the specified string to the end of this string.
6	**boolean contentEquals(StringBuffer sb)** Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	**static String copyValueOf(char[] data)** Returns a String that represents the character sequence in the array specified.
8	**static String copyValueOf(char[] data, int offset, int count)** Returns a String that represents the character sequence in the array specified.
9	**boolean endsWith(String suffix)** Tests if this string ends with the specified suffix.
10	**boolean equals(Object anObject)** Compares this string to the specified object.
11	**boolean equalsIgnoreCase(String anotherString)** Compares this String to another String, ignoring case considerations.
12	**byte getBytes()** Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	**byte[] getBytes(String charsetName)** Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** Copies characters from this string into the destination character array.

15	**int hashCode()** Returns a hash code for this string.
16	**int indexOf(int ch)** Returns the index within this string of the first occurrence of the specified character.
17	**int indexOf(int ch, int fromIndex)** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
18	**int indexOf(String str)** Returns the index within this string of the first occurrence of the specified substring.
19	**int indexOf(String str, int fromIndex)** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20	**String intern()** Returns a canonical representation for the string object.
21	**int lastIndexOf(int ch)** Returns the index within this string of the last occurrence of the specified character.
22	**int lastIndexOf(int ch, int fromIndex)** Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
23	**int lastIndexOf(String str)** Returns the index within this string of the rightmost occurrence of the specified substring.
24	**int lastIndexOf(String str, int fromIndex)** Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
25	**int length()** Returns the length of this string.
26	**boolean matches(String regex)** Tells whether or not this string matches the given regular expression.
27	**boolean regionMatches(boolean ignoreCase, int toffset, String other, int offset, int len)** Tests if two string regions are equal.
28	**boolean regionMatches(int toffset, String other, int offset, int len)** Tests if two string regions are equal.
29	**String replace(char *oldChar*, char newChar)**

	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
30	**String replaceAll(String regex, String replacement** Replaces each substring of this string that matches the given regular expression with the given replacement.
31	**String replaceFirst(String regex, String replacement)** Replaces the first substring of this string that matches the given regular expression with the given replacement.
32	**String[] split(String regex)** Splits this string around matches of the given regular expression.
33	**String[] split(String regex, int limit)** Splits this string around matches of the given regular expression.
34	**boolean startsWith(String prefix)** Tests if this string starts with the specified prefix.
35	**boolean startsWith(String prefix, int toffset)** Tests if this string starts with the specified prefix beginning a specified index.
36	**CharSequence subSequence(int beginIndex, int endIndex)** Returns a new character sequence that is a subsequence of this sequence.
37	**String substring(int beginIndex)** Returns a new string that is a substring of this string.
38	**String substring(int beginIndex, int endIndex)** Returns a new string that is a substring of this string.
39	**char[] toCharArray()** Converts this string to a new character array.
40	**String toLowerCase()** Converts all of the characters in this String to lower case using the rules of the default locale.
41	**String toLowerCase(Locale locale)** Converts all of the characters in this String to lower case using the rules of the given Locale.
42	**String toString()** This object (which is already a string!) is itself returned.
43	**String toUpperCase()** Converts all of the characters in this String to upper case using the rules of the default locale.

44	**String toUpperCase(Locale locale)** Converts all of the characters in this String to upper case using the rules of the given Locale.
45	**String trim()** Returns a copy of the string, with leading and trailing whitespace omitted.
46	**static String valueOf(primitive data type x)** Returns the string representation of the passed data type argument.

Array Methods

Sr.No	Methods with Description
1	**def apply(x: T, xs: T*): Array[T]** Creates an array of T objects, where T can be Unit, Double, Float, Long, Int, Char, Short, Byte, Boolean.
2	**def concat[T](xss: Array[T]*): Array[T]** Concatenates all arrays into a single array.
3	**def copy(src: AnyRef, srcPos: Int, dest: AnyRef, destPos: Int, length: Int): Unit** Copy one array to another. Equivalent to Java's System.arraycopy(src, srcPos, dest, destPos, length).
4	**def empty[T]: Array[T]** Returns an array of length 0
5	**def iterate[T](start: T, len: Int)(f: (T) => T): Array[T]** Returns an array containing repeated applications of a function to a start value.
6	**def fill[T](n: Int)(elem: => T): Array[T]** Returns an array that contains the results of some element computation a number of times.
7	**def fill[T](n1: Int, n2: Int)(elem: => T): Array[Array[T]]** Returns a two-dimensional array that contains the results of some element computation a number of times.
8	**def iterate[T](start: T, len: Int)(f: (T) => T): Array[T]** Returns an array containing repeated applications of a function to a start value.
9	**def ofDim[T](n1: Int): Array[T]**

	Creates array with given dimensions.
10	**def ofDim[T](n1: Int, n2: Int): Array[Array[T]]** Creates a 2-dimensional array
11	**def ofDim[T](n1: Int, n2: Int, n3: Int): Array[Array[Array[T]]]** Creates a 3-dimensional array
12	**def range(start: Int, end: Int, step: Int): Array[Int]** Returns an array containing equally spaced values in some integer interval.
13	**def range(start: Int, end: Int): Array[Int]** Returns an array containing a sequence of increasing integers in a range.
14	**def tabulate[T](n: Int)(f: (Int)=> T): Array[T]** Returns an array containing values of a given function over a range of integer values starting from 0.
15	**def tabulate[T](n1: Int, n2: Int)(f: (Int, Int) => T): Array[Array[T]]** Returns a two-dimensional array containing values of a given function over ranges of integer values starting from 0.

List Methods

Sr.No	Methods with Description
1	**def +(elem: A): List[A]** **Prepends an element to this list**
2	**def ::(x: A): List[A]** **Adds an element at the beginning of this list.**
3	**def :::(prefix: List[A]): List[A]** **Adds the elements of a given list in front of this list.**

4	**def ::(x: A): List[A]** **Adds an element x at the beginning of the list**
5	**def addString(b: StringBuilder): StringBuilder** **Appends all elements of the list to a string builder.**
6	**def addString(b: StringBuilder, sep: String): StringBuilder** **Appends all elements of the list to a string builder using a separator string.**
7	**def apply(n: Int): A** **Selects an element by its index in the list.**
8	**def contains(elem: Any): Boolean** **Tests whether the list contains a given value as an element.**
9	**def copyToArray(xs: Array[A], start: Int, len: Int): Unit** **Copies elements of the list to an array. Fills the given array xs with at most length (len) elements of this list, beginning at position start.**
10	**def distinct: List[A]** **Builds a new list from the list without any duplicate elements.**
11	**def drop(n: Int): List[A]** **Returns all elements except first n ones.**
12	**def dropRight(n: Int): List[A]** **Returns all elements except last n ones.**
13	**def dropWhile(p: (A) => Boolean): List[A]** **Drops longest prefix of elements that satisfy a predicate.**
14	**def endsWith[B](that: Seq[B]): Boolean**

	Tests whether the list ends with the given sequence.
15	**def equals(that: Any): Boolean** **The equals method for arbitrary sequences. Compares this sequence to some other object.**
16	**def exists(p: (A) => Boolean): Boolean** **Tests whether a predicate holds for some of the elements of the list.**
17	**def filter(p: (A) => Boolean): List[A]** **Returns all elements of the list which satisfy a predicate.**
18	**def forall(p: (A) => Boolean): Boolean** **Tests whether a predicate holds for all elements of the list.**
19	**def foreach(f: (A) => Unit): Unit** **Applies a function f to all elements of the list.**
20	**def head: A** **Selects the first element of the list.**
21	**def indexOf(elem: A, from: Int): Int** **Finds index of first occurrence value in the list, after the index position.**
22	**def init: List[A]** **Returns all elements except the last.**
23	**def intersect(that: Seq[A]): List[A]** **Computes the multiset intersection between the list and another sequence.**
24	**def isEmpty: Boolean**

	Tests whether the list is empty.
25	**def iterator: Iterator[A]** **Creates a new iterator over all elements contained in the iterable object.**
26	**def last: A** **Returns the last element.**
27	**def lastIndexOf(elem: A, end: Int): Int** **Finds index of last occurrence of some value in the list; before or at a given end index.**
28	**def length: Int** **Returns the length of the list.**
29	**def map[B](f: (A) => B): List[B]** **Builds a new collection by applying a function to all elements of this list.**
30	**def max: A** **Finds the largest element.**
31	**def min: A** **Finds the smallest element.**
32	**def mkString: String** **Displays all elements of the list in a string.**
33	**def mkString(sep: String): String** **Displays all elements of the list in a string using a separator string.**
34	**def reverse: List[A]**

	Returns new list with elements in reversed order.
35	**def sorted[B >: A]: List[A]** **Sorts the list according to an Ordering.**
36	**def startsWith[B](that: Seq[B], offset: Int): Boolean** **Tests whether the list contains the given sequence at a given index.**
37	**def sum: A** **Sums up the elements of this collection.**
38	**def tail: List[A]** **Returns all elements except the first.**
39	**def take(n: Int): List[A]** **Returns first "n" elements.**
40	**def takeRight(n: Int): List[A]** **Returns last "n" elements.**
41	**def toArray: Array[A]** **Converts the list to an array.**
42	**def toBuffer[B >: A]: Buffer[B]** **Converts the list to a mutable buffer.**
43	**def toMap[T, U]: Map[T, U]** **Converts this list to a map.**
44	**def toSeq: Seq[A]** **Converts the list to a sequence.**
45	**def toSet[B >: A]: Set[B]**

	Converts the list to a set.
46	**def toString(): String** **Converts the list to a string.**

SET Methods

Sr.No	Methods with Description
1	**def +(elem: A): Set[A]** Creates a new set with an additional element, unless the element is already present.
2	**def -(elem: A): Set[A]** Creates a new set with a given element removed from this set.
3	**def contains(elem: A): Boolean** Returns true if elem is contained in this set, false otherwise.
4	**def &(that: Set[A]): Set[A]** Returns a new set consisting of all elements that are both in this set and in the given set.
5	**def &~(that: Set[A]): Set[A]** Returns the difference of this set and another set.
6	**def +(elem1: A, elem2: A, elems: A*): Set[A]** Creates a new immutable set with additional elements from the passed sets
7	**def ++(elems: A): Set[A]** Concatenates this immutable set with the elements of another collection to this immutable set.
8	**def -(elem1: A, elem2: A, elems: A*): Set[A]** Returns a new immutable set that contains all elements of the current immutable set except one less occurrence of each of the given argument elements.
9	**def addString(b: StringBuilder): StringBuilder** Appends all elements of this immutable set to a string builder.

10	**def addString(b: StringBuilder, sep: String): StringBuilder** Appends all elements of this immutable set to a string builder using a separator string.
11	**def apply(elem: A)** Tests if some element is contained in this set.
12	**def count(p: (A) => Boolean): Int** Counts the number of elements in the immutable set which satisfy a predicate.
13	**def copyToArray(xs: Array[A], start: Int, len: Int): Unit** Copies elements of this immutable set to an array.
14	**def diff(that: Set[A]): Set[A]** Computes the difference of this set and another set.
15	**def drop(n: Int): Set[A]]** Returns all elements except first n ones.
16	**def dropRight(n: Int): Set[A]** Returns all elements except last n ones.
17	**def dropWhile(p: (A) => Boolean): Set[A]** Drops longest prefix of elements that satisfy a predicate.
18	**def equals(that: Any): Boolean** The equals method for arbitrary sequences. Compares this sequence to some other object.
19	**def exists(p: (A) => Boolean): Boolean** Tests whether a predicate holds for some of the elements of this immutable set.
20	**def filter(p: (A) => Boolean): Set[A]** Returns all elements of this immutable set which satisfy a predicate.
21	**def find(p: (A) => Boolean): Option[A]** Finds the first element of the immutable set satisfying a predicate, if any.
22	**def forall(p: (A) => Boolean): Boolean** Tests whether a predicate holds for all elements of this immutable set.
23	**def foreach(f: (A) => Unit): Unit** Applies a function f to all elements of this immutable set.
24	**def head: A**

	Returns the first element of this immutable set.
25	**def init: Set[A]** Returns all elements except the last.
26	**def intersect(that: Set[A]): Set[A]** Computes the intersection between this set and another set.
27	**def isEmpty: Boolean** Tests if this set is empty.
28	**def iterator: Iterator[A]** Creates a new iterator over all elements contained in the iterable object.
29	**def last: A** Returns the last element.
30	**def map[B](f: (A) => B): immutable.Set[B]** Builds a new collection by applying a function to all elements of this immutable set.
31	**def max: A** Finds the largest element.
32	**def min: A** Finds the smallest element.
33	**def mkString: String** Displays all elements of this immutable set in a string.
34	**def mkString(sep: String): String** Displays all elements of this immutable set in a string using a separator string.
35	**def product: A** Returns the product of all elements of this immutable set with respect to the * operator in num.
36	**def size: Int** Returns the number of elements in this immutable set.
37	**def splitAt(n: Int): (Set[A], Set[A])** Returns a pair of immutable sets consisting of the first n elements of this immutable set, and the other elements.
38	**def subsetOf(that: Set[A]): Boolean** Returns true if this set is a subset of that, i.e. if every element of this set is also an element of that.

39	**def sum: A** Returns the sum of all elements of this immutable set with respect to the + operator in num.
40	**def tail: Set[A]** Returns a immutable set consisting of all elements of this immutable set except the first one.
41	**def take(n: Int): Set[A]** Returns first n elements.
42	**def takeRight(n: Int):Set[A]** Returns last n elements.
43	**def toArray: Array[A]** Returns an array containing all elements of this immutable set.
44	**def toBuffer[B >: A]: Buffer[B]** Returns a buffer containing all elements of this immutable set.
45	**def toList: List[A]** Returns a list containing all elements of this immutable set.
46	**def toMap[T, U]: Map[T, U]** Converts this immutable set to a map
47	**def toSeq: Seq[A]** Returns a seq containing all elements of this immutable set.
48	**def toString(): String** Returns a String representation of the object.

Map Methods

Sr.No	Methods with Description
1	**def ++(xs: Map[(A, B)]): Map[A, B]** **Returns a new map containing mappings of this map and those provided by xs.**
2	**def -(elem1: A, elem2: A, elems: A*): Map[A, B]** **Returns a new map containing all the mappings of this map except mappings with a key equal to elem1, elem2 or any of elems.**
3	**def --(xs: GTO[A]): Map[A, B]** **Returns a new map with all the key/value mappings of this map except mappings with a key equal to a key from the traversable object xs.**
4	**def get(key: A): Option[B]** **Optionally returns the value associated with a key.**
5	**def iterator: Iterator[(A, B)]** **Creates a new iterator over all key/value pairs of this map**
6	**def addString(b: StringBuilder): StringBuilder** **Appends all elements of this shrinkable collection to a string builder.**
7	**def addString(b: StringBuilder, sep: String): StringBuilder** **Appends all elements of this shrinkable collection to a string builder using a separator string.**
8	**def apply(key: A): B** **Returns the value associated with the given key, or the result of the map's default method, if none exists.**

9	**def clear(): Unit** **Removes all bindings from the map. After this operation has completed, the map will be empty.**
10	**def clone(): Map[A, B]** **Creates a copy of the receiver object.**
11	**def contains(key: A): Boolean** **Returns true if there is a binding for key in this map, false otherwise.**
12	**def copyToArray(xs: Array[(A, B)]): Unit** **Copies values of this shrinkable collection to an array. Fills the given array xs with values of this shrinkable collection.**
13	**def count(p: ((A, B)) => Boolean): Int** **Counts the number of elements in the shrinkable collection which satisfy a predicate.**
14	**def default(key: A): B** **Defines the default value computation for the map, returned when a key is not found.**
15	**def drop(n: Int): Map[A, B]** **Returns all elements except first n ones.**
16	**def dropRight(n: Int): Map[A, B]** **Returns all elements except last n ones**
17	**def dropWhile(p: ((A, B)) => Boolean): Map[A, B]** **Drops longest prefix of elements that satisfy a predicate.**
18	**def empty: Map[A, B]** **Returns the empty map of the same type.**

19	**def equals(that: Any): Boolean** **Returns true if both maps contain exactly the same keys/values, false otherwise.**
20	**def exists(p: ((A, B)) => Boolean): Boolean** **Returns true if the given predicate p holds for some of the elements of this shrinkable collection, otherwise false.**
21	**def filter(p: ((A, B))=> Boolean): Map[A, B]** **Returns all elements of this shrinkable collection which satisfy a predicate.**
22	**def filterKeys(p: (A) => Boolean): Map[A, B]** **Returns an immutable map consisting only of those key value pairs of this map where the key satisfies the predicate p.**
23	**def find(p: ((A, B)) => Boolean): Option[(A, B)]** **Finds the first element of the shrinkable collection satisfying a predicate, if any.**
24	**def foreach(f: ((A, B)) => Unit): Unit** **Applies a function f to all elements of this shrinkable collection.**
25	**def init: Map[A, B]** **Returns all elements except the last.**
26	**def isEmpty: Boolean** **Tests whether the map is empty.**
27	**def keys: Iterable[A]** **Returns an iterator over all keys.**
28	**def last: (A, B)** **Returns the last element.**

29	**def max: (A, B)** **Finds the largest element.**
30	**def min: (A, B)** **Finds the smallest element.**
31	**def mkString: String** **Displays all elements of this shrinkable collection in a string.**
32	**def product: (A, B)** **Returns the product of all elements of this shrinkable collection with respect to the * operator in num.**
33	**def remove(key: A): Option[B]** **Removes a key from this map, returning the value associated previously with that key as an option.**
34	**def retain(p: (A, B) => Boolean): Map.this.type** **Retains only those mappings for which the predicate p returns true.**
35	**def size: Int** **Return the number of elements in this map.**
36	**def sum: (A, B)** **Returns the sum of all elements of this shrinkable collection with respect to the + operator in num.**
37	**def tail: Map[A, B]** **Returns all elements except the first.**
38	**def take(n: Int): Map[A, B]** **Returns first n elements.**

39	**def takeRight(n: Int): Map[A, B]** **Returns last n elements.**
40	**def takeWhile(p: ((A, B)) => Boolean): Map[A, B]** **Takes longest prefix of elements that satisfy a predicate.**
41	**def toArray: Array[(A, B)]** **Converts this shrinkable collection to an array.**
42	**def toBuffer[B >: A]: Buffer[B]** **Returns a buffer containing all elements of this map.**
43	**def toList: List[A]** **Returns a list containing all elements of this map.**
44	**def toSeq: Seq[A]** **Returns a seq containing all elements of this map.**
45	**def toSet: Set[A]** **Returns a set containing all elements of this map.**
46	**def toString(): String** **Returns a String representation of the object.**

Iterator methods

Sr.No	Methods with Description
1	**def hasNext: Boolean** **Tests whether this iterator can provide another element.**
2	**def next(): A** **Produces the next element of this iterator.**
3	**def ++(that: => Iterator[A]): Iterator[A]** **Concatenates this iterator with another.**
4	**def ++[B >: A](that :=> GenTraversableOnce[B]): Iterator[B]** **Concatenates this iterator with another.**
5	**def addString(b: StringBuilder): StringBuilder** **Returns the string builder b to which elements were appended.**
6	**def addString(b: StringBuilder, sep: String): StringBuilder** **Returns the string builder b to which elements were appended using a separator string.**
7	**def buffered: BufferedIterator[A]** **Creates a buffered iterator from this iterator.**
8	**def contains(elem: Any): Boolean** **Tests whether this iterator contains a given value as an element.**
9	**def copyToArray(xs: Array[A], start: Int, len: Int): Unit** **Copies selected values produced by this iterator to an array.**

10	**def count(p: (A) => Boolean): Int** **Counts the number of elements in the traversable or iterator which satisfy a predicate.**
11	**def drop(n: Int): Iterator[A]** **Advances this iterator past the first n elements, or the length of the iterator, whichever is smaller.**
12	**def dropWhile(p: (A) => Boolean): Iterator[A]** **Skips longest sequence of elements of this iterator which satisfy given predicate p, and returns an iterator of the remaining elements.**
13	**def duplicate: (Iterator[A], Iterator[A])** **Creates two new iterators that both iterate over the same elements as this iterator (in the same order).**
14	**def exists(p: (A) => Boolean): Boolean** **Returns true if the given predicate p holds for some of the values produced by this iterator, otherwise false.**
15	**def filter(p: (A) => Boolean): Iterator[A]** **Returns an iterator over all the elements of this iterator that satisfy the predicate p. The order of the elements is preserved.**
16	**def filterNot(p: (A) => Boolean): Iterator[A]** **Creates an iterator over all the elements of this iterator which do not satisfy a predicate p.**
17	**def find(p: (A) => Boolean): Option[A]** **Finds the first value produced by the iterator satisfying a predicate, if any.**
18	**def flatMap[B](f: (A) => GenTraversableOnce[B]): Iterator[B]**

	Creates a new iterator by applying a function to all values produced by this iterator and concatenating the results.
19	**def forall(p: (A) => Boolean): Boolean** **Returns true if the given predicate p holds for all values produced by this iterator, otherwise false.**
20	**def foreach(f: (A) => Unit): Unit** **Applies a function f to all values produced by this iterator.**
21	**def hasDefiniteSize: Boolean** **Returns true for empty Iterators, false otherwise.**
22	**def indexOf(elem: B): Int** **Returns the index of the first occurrence of the specified object in this iterable object.**
23	**def indexWhere(p: (A) => Boolean): Int** **Returns the index of the first produced value satisfying a predicate, or -1.**
24	**def isEmpty: Boolean** **Returns true if hasNext is false, false otherwise.**
25	**def isTraversableAgain: Boolean** **Tests whether this Iterator can be repeatedly traversed.**
26	**def length: Int** **Returns the number of elements in this iterator. The iterator is at its end after this method returns.**
27	**def map[B](f: (A) => B): Iterator[B]** **Returns a new iterator which transforms every value produced by this iterator by applying the function f to it.**

28	**def max: A** **Finds the largest element. The iterator is at its end after this method returns.**
29	**def min: A** **Finds the minimum element. The iterator is at its end after this method returns.**
30	**def mkString: String** **Displays all elements of this traversable or iterator in a string.**
31	**def mkString(sep: String): String** **Displays all elements of this traversable or iterator in a string using a separator string.**
32	**def nonEmpty: Boolean** **Tests whether the traversable or iterator is not empty.**
33	**def padTo(len: Int, elem: A): Iterator[A]** **Appends an element value to this iterator until a given target length is reached.**
34	**def patch(from: Int, patchElems: Iterator[B], replaced: Int): Iterator[B]** **Returns this iterator with patched values.**
35	**def product: A** **Multiplies up the elements of this collection.**
36	**def sameElements(that: Iterator[_]): Boolean** **Returns true, if both iterators produce the same elements in the same order, false otherwise.**
37	**def seq: Iterator[A]** **Returns a sequential view of the collection.**

38	**def size: Int** **Returns the number of elements in this traversable or iterator.**
39	**def slice(from: Int, until: Int): Iterator[A]** **Creates an iterator returning an interval of the values produced by this iterator.**
40	**def sum: A** **Returns the sum of all elements of this traversable or iterator with respect to the + operator in num.**
41	**def take(n: Int): Iterator[A]** **Returns an iterator producing only of the first n values of this iterator, or else the whole iterator, if it produces fewer than n values.**
42	**def toArray: Array[A]** **Returns an array containing all elements of this traversable or iterator.**
43	**def toBuffer: Buffer[B]** **Returns a buffer containing all elements of this traversable or iterator.**
44	**def toIterable: Iterable[A]** **Returns an Iterable containing all elements of this traversable or iterator. This will not terminate for infinite iterators.**
45	**def toIterator: Iterator[A]** **Returns an Iterator containing all elements of this traversable or iterator. This will not terminate for infinite iterators.**
46	**def toList: List[A]** **Returns a list containing all elements of this traversable or iterator.**

47	**def toMap[T, U]: Map[T, U]** **Returns a map containing all elements of this traversable or iterator.**
48	**def toSeq: Seq[A]** **Returns a sequence containing all elements of this traversable or iterator.**
49	**def toString(): String** **Converts this iterator to a string.**
50	**def zip[B](that: Iterator[B]): Iterator[(A, B)** **Returns a new iterator containing pairs consisting of corresponding elements of this iterator. The number of elements returned by the new iterator is same as the minimum number of elements returned by the iterator (A or B).**

Options methods

Sr.No	Methods with Description
1	**def get: A** **Returns the option's value.**
2	**def isEmpty: Boolean** **Returns true if the option is None, false otherwise.**
3	**def productArity: Int** **The size of this product. For a product A(x_1, ..., x_k), returns k**

4	**def productElement(n: Int): Any** **The nth element of this product, 0-based. In other words, for a product A(x_1, ..., x_k), returns x_(n+1) where 0 < n < k.**
5	**def exists(p: (A) => Boolean): Boolean** **Returns true if this option is nonempty and the predicate p returns true when applied to this Option's value. Otherwise, returns false.**
6	**def filter(p: (A) => Boolean): Option[A]** **Returns this Option if it is nonempty and applying the predicate p to this Option's value returns true. Otherwise, return None.**
7	**def filterNot(p: (A) => Boolean): Option[A]** **Returns this Option if it is nonempty and applying the predicate p to this Option's value returns false. Otherwise, return None.**
8	**def flatMap[B](f: (A) => Option[B]): Option[B]** **Returns the result of applying f to this Option's value if this Option is nonempty. Returns None if this Option is empty.**
9	**def foreach[U](f: (A) => U): Unit** **Apply the given procedure f to the option's value, if it is nonempty. Otherwise, do nothing.**
10	**def getOrElse[B >: A](default: => B): B** **Returns the option's value if the option is nonempty, otherwise return the result of evaluating default.**
11	**def isDefined: Boolean** **Returns true if the option is an instance of Some, false otherwise.**
12	**def iterator: Iterator[A]**

	Returns a singleton iterator returning the Option's value if it is nonempty, or an empty iterator if the option is empty.
13	**def map[B](f: (A) => B): Option[B]** **Returns a Some containing the result of applying f to this Option's value if this Option is nonempty. Otherwise return None.**
14	**def orElse[B >: A](alternative: => Option[B]): Option[B]** **Returns this Option if it is nonempty, otherwise return the result of evaluating alternative.**
15	**def orNull** **Returns the option's value if it is nonempty, or null if it is empty.**