

**Function** is a category of functional interfaces that takes an object of type **T** and returns an object of type **R**.

Until now, the functional interfaces that we’ve discussed have either not taken any argument(**Supplier**), not returned any value(**Consumer**), or returned only a boolean(**Predicate**).

**Function** interfaces are very useful as we can specify the type of input and output.

Below are some of the interfaces that fall in this category.

Interface	Purpose	Abstract Method
Function<T,R>	Represents a function that accepts one argument and produces a result	R apply (T t)
DoubleFunction<R>	Accepts a double-valued argument and produces a result	R apply (double t)
IntFunction<R>	Accepts an int-valued argument and produces a result	R apply (int t)
LongFunction<R>	Accepts a long-valued argument and produces a result	R apply (long t)
DoubleToIntFunction	Accepts a double-valued argument and produces an int-valued result	int applyAsInt(double value)
DoubleToLongFunction	Accepts a double-valued argument and produces a long-valued result	long applyAsLong(double value)
IntToDoubleFunction	Accepts an int-valued argument and produces a double-valued result	double applyAsDouble(int value)
IntToLongFunction	Accepts an int-valued argument and produces a long-valued result	long applyAsLong(int value)
LongToIntFunction	Accepts a long-valued argument and produces an int-valued result	int applyAsInt(long value)
LongToDoubleFunction	Accepts a long-valued argument and produces a double-valued result.	double applyAsDouble(long value)
ToDoubleFunction<T>	Accepts a reference type and produces an int-valued result	double applyAsDouble(T value)
ToIntFunction<T>	Accepts a reference type and produces an int-valued result	int applyAsInt(T value)
ToLongFunction<T>	Accepts a reference type and produces a long-valued result.	long applyAsLong(T value)
BiFunction<T,U,R>	Represents a function that accepts two arguments and produces a result	R apply(T t, U u)
ToDoubleBiFunction<T,U>	Accepts two reference type arguments and produces a double-valued result	R applyAsDouble(T t, U u)
ToIntBiFunction<T,U>	Accepts two reference type arguments and produces an int-valued result	R applyAsInt(T t, U u)
ToLongBiFunction<T,U>	Accepts two reference type arguments and produces a long-valued result	R applyAsLong(T t, U u)

Let us discuss some types of **Function** functional interfaces.

## Function<T, R>#

The function takes only one argument of type **T** and returns a result of type **R**.

The following is the list of all the methods in the **Function<T, R>** interface.

Method Summary

All Methods	Static Methods	Instance Methods	Abstract Methods	Default Methods
Modifier and Type		Method and Description		
default	<V>	Function<T,V>	andThen(Function<? super R,? extends V> after) Returns a composed function that first applies this function to its input, and then applies the <b>after</b> function to the result.	
		R	apply(T t) Applies this function to the given argument.	
default	<V>	Function<V,R>	compose(Function<? super V,? extends T> before) Returns a composed function that first applies the <b>before</b> function to its input, and then applies this function to the result.	
static	<T>	Function<T,T>	identity() Returns a function that always returns its input argument.	

Let’s look at an example of each method:

## R apply(T t)#

This is the abstract method of the **Function** interface. It takes one argument of type **T** as input and returns a value of type **R**.

In the below example, we will create a function called **lengthFunction** . It takes a string as input and returns the length of the string as output.

```
1 import java.util.function.Function;
2
3 public class FunctionInterfaceDemo {
4
5     public static void main(String[] args) {
6         // Created a function which returns the length of string.
7         Function<String, Integer> lengthFunction = str -> str.length();
8
9         System.out.println("String length: " + lengthFunction.apply("This is awesome!!"));
10    }
11 }
12 }
```

Run Save Reset

## compose(Function<? super V, ? extends T> before)#

Returns a composed function that first applies the function provided as a parameter on the input, and then applies the function on which it is called, to the result.

```
1 import java.util.function.Function;
2
3 public class FunctionDemo {
4
5     public static void main(String args[]) {
6
7         // Function which adds 10 to the given element.
8         Function<Integer, Integer> increment = x -> x + 10;
9         // Function which doubles the given element.
10        Function<Integer, Integer> multiply = y -> y * 2;
11        // Since we are using compose(), multiplication will be done first and then increment will be done
12        System.out.println("compose result: " + increment.compose(multiply).apply(3));
13    }
14 }
15 }
16 }
```

Run Save Reset

## andThen(Function<? super R,? extends V> after)#

This method returns a composed function that first applies the function on which it is called on the input, and then applies the function provided as parameter, to the result.

```
1 import java.util.function.Function;
2
3 public class FunctionDemo {
4
5     public static void main(String args[]) {
6         Function<Integer,Integer> increment = x -> x + 10;
7         Function<Integer,Integer> multiply = y -> y * 2;
8         // Since we are using andThen(), increment will be done first and then multiplication will be done
9         System.out.println("andThen result: " + increment.andThen(multiply).apply(3));
10    }
11 }
12 }
13 }
```

Run Save Reset

## BiFunction<T,U,R>#

The **BiFunction<T, U, R>** is similar to **Function<T, R>** interface; the only difference is that the **BiFunction** interface takes in two parameters and returns an output.

Below is the list of methods in the **BiFunction** interface.

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods
Modifier and Type		Method and Description	
default	<V> BiFunction<T,U,V>	andThen(Function<? super R,? extends V> after) Returns a composed function that first applies this function to its input, and then applies the <b>after</b> function to the result.	
R		apply(T t, U u) Applies this function to the given arguments.	

In the below example, we will create a **BiFunction** that takes two numbers as input and returns their sum.

```
1 import java.util.function.BiFunction;
2
3 public class BiFunctionInterfaceDemo {
4     public static void main(String args[])
5     {
6
7         BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;
8
9         System.out.println("Sum = " + add.apply(2, 3));
10    }
11 }
12 }
```