ScalaFunctions

January 2, 2025

[1]:
```
// Basic function to greet a user
def greet(name: String, city: String): String = {
  s"Welcome $name to $city!"
}
println(greet("Navadeep", "Hyderabad"))
```

```
Welcome Navadeep to Hyderabad!
```

```
greet: (name: String, city: String)String
```

[2]:
```
// Function with default parameters
def introduce(name: String = "Guest", age: Int = 25): String = {
  s"Hi, I am $name, and I am $age years old."
}
println(introduce("Navadeep", 28))
println(introduce("Sameera"))
println(introduce())
```

```
introduce: (name: String, age: Int)String
```

```
Hi, I am Navadeep, and I am 28 years old.
Hi, I am Sameera, and I am 25 years old.
Hi, I am Guest, and I am 25 years old.
```

[3]:
```
// Lambda function for addition
val add = (a: Int, b: Int) => a + b
println(s"Sum: ${add(10, 20)}")

// Lambda function for string formatting
val formatString = (s: String) => s.toUpperCase()
println(formatString("scala is amazing"))
```

```
Sum: 30
SCALA IS AMAZING
```

```
add = > Int = $Lambda$2081/0x0000000100d4d040@771839ea
formatString = > String = $Lambda$2082/0x0000000100d4e040@6151c74e
```

[3]: > String = $Lambda$2082/0x0000000100d4e040@6151c74e

[4]:
```
// Higher-order function
def performOperation(x: Int, y: Int, operation: (Int, Int) => Int): Int = {
  operation(x, y)
}

val multiply = (a: Int, b: Int) => a * b
println(s"Multiplication: ${performOperation(5, 4, multiply)}")
```

```
Multiplication: 20
```

```
performOperation: (x: Int, y: Int, operation: (Int, Int) => Int)Int
multiply = > Int = $Lambda$2096/0x0000000100d5d040@371a60ee
```

[4]: > Int = $Lambda$2096/0x0000000100d5d040@371a60ee

[5]:
```
// Function stored in a variable
val square: Int => Int = x => x * x
println(s"Square of 6: ${square(6)}")

// Passing a function as an argument
def applyFunction(value: Int, func: Int => Int): Int = func(value)
println(s"Applied Function Result: ${applyFunction(3, square)}")
```

```
Square of 6: 36
Applied Function Result: 9
```

```
square = > Int = $Lambda$2098/0x0000000100d5f040@64842bb0
applyFunction: (value: Int, func: Int => Int)Int
```

[5]: > Int = $Lambda$2098/0x0000000100d5f040@64842bb0

[6]:
```
// Curried function example
def curriedAdder(a: Int)(b: Int): Int = a + b
val addFive = curriedAdder(5) _
println(s"Result after adding 5: ${addFive(10)}")
```

```
Result after adding 5: 15
```

```
addFive = > Int = $Lambda$2105/0x0000000100d6b040@4a3a439


curriedAdder: (a: Int)(b: Int)Int
```

[6]: > Int = $Lambda$2105/0x0000000100d6b040@4a3a439

[7]:
```
// Recursive function to calculate factorial
def factorial(n: Int): Int = {
  if (n == 0) 1
  else n * factorial(n - 1)
}
println(s"Factorial of 5: ${factorial(5)}")
```

```
factorial: (n: Int)Int


Factorial of 5: 120
```

[8]:
```
// Function to filter even numbers
def filterEvens(nums: Array[Int], condition: Int => Boolean): Array[Int] = {
  nums.filter(condition)
}

val numbers = Array(1, 2, 3, 4, 5, 6)
val evens = filterEvens(numbers, _ % 2 == 0)
println(s"Even Numbers: ${evens.mkString(", ")}")
```

```
Even Numbers: 2, 4, 6
```

```
filterEvens: (nums: Array[Int], condition: Int => Boolean)Array
numbers = Array(1, 2, 3, 4, 5, 6)
evens = Array(2, 4, 6)
```

[8]:
```
Array(2, 4, 6)
```

[9]:
```
// Function composition example
val increment: Int => Int = _ + 1
val double: Int => Int = _ * 2
val incrementAndDouble = increment.andThen(double)

println(s"Result after composition: ${incrementAndDouble(3)}")
```

```
Result after composition: 8
```

```
increment = > Int = $Lambda$2161/0x0000000100da2040@5bb5ec8d
double = > Int = $Lambda$2162/0x0000000100da2840@7b70d847
incrementAndDouble = > Int = scala.Function1$$Lambda$2163/
  ↪0x0000000100da3840@493af50c
```

[9]:
```
> Int = scala.Function1$$Lambda$2163/0x0000000100da3840@493af50c
```

[ ]:
```

```