Partially Applied Functions

Partially applied functions are functions that have some of their arguments fixed in advance. This means you create a new function by filling in one or more arguments of an existing function.

For example, if you have a function that adds three numbers:

```
def add(x: Int, y: Int, z: Int): Int = x + y + z
```

You can create a partially applied function that adds 10 to any two numbers like this:

```
val add10 = add(10, _: Int, _: Int)
```

Now, `add10` is a function that takes two integers and adds 10 to their sum.

Partial Functions

Partial functions are functions that are not defined for all possible inputs. They only work for certain values.

For instance:

```
val isEven: PartialFunction[Int, String] = {
  case x if x % 2 == 0 => s"$x is even"
}
```

This `isEven` function only works for even numbers. If you pass it an odd number, it will throw an exception unless you handle it with something like `isDefinedAt`.

Implicit Conversion / Implicit Types

Implicit conversions allow you to automatically convert one type to another. This can make code more readable and reduce boilerplate.

For example, you can define an implicit conversion from `String` to `Int`:

```
implicit def stringToInt(s: String): Int = s.toInt
```

Now, you can pass a `String` where an `Int` is expected, and the compiler will automatically convert it.

Implicit types work similarly. They allow you to define default types for parameters that can be automatically filled in by the compiler if not provided explicitly.

Underscore Syntax in Collection Operations

The underscore (`_`) is used in Scala for various purposes, including as a placeholder in collection operations.

For example, in a `map` operation:

```
val nums = List(1, 2, 3, 4)
val doubled = nums.map(_ * 2)
```

Here, `_ * 2` is shorthand for `x => x * 2`. The underscore represents each element in the collection, making the code concise.

Similarly, you can use underscores in other collection operations like `filter`:

```
val evens = nums.filter(_ % 2 == 0)
```

This filters the list to only include even numbers, with `_ % 2 == 0` being shorthand for `x => x % 2 == 0`.