Syntax [edit]

Functional programming style [edit]

Kotlin relaxes Java's restriction of allowing staticmethods and variables to exist only within a class body. Static objects and functions can be defined at the top level of the package without needing a redundant class level. For compatibility with Java, Kotlin provides a `JvmName` annotation which specifies a class name used when the package is viewed from a Java project. For example, `@file:JvmName("JavaClassName")`.

Main entry point [edit]

As in C and C++, the entry pointto a Kotlin programis a function named "main", which is passed an array containing any command linearguments. Perland Unix/Linux shell script-style string interpolationis supported. Type inferenceis also supported.

```
1 // Hello, World! example
2 fun main(args: Array<String>) {
3   val scope = "World"
4   println("Hello, $scope!")
5 }
```

Extension methods [edit]

Similar to C#, Kotlin allows a user to add methods to any class without the formalities of creating a derived class with new methods. Instead, Kotlin adds the concept of an extension method which allows a function to be "glued" onto the public method list of any class without being formally placed inside of the class. In other words, an extension method is a helper method that has access to all the public interface of a class which it can use to create a new method interface to a target class and this method will appear exactly like a method of the class, appearing as part of code completion inspection of class methods. For example:

```
1     package MyStringExtensions
2
3     fun String.lastChar(): Char = get(length - 1)
4
5     >>> println("Kotlin".lastChar())
```

By placing the preceding code in the top-level of a package, the String class is extended to include a lastChar method that was not included in the original definition of the String class.

```
1     // overloading '+' operator using an extension method
2     operator fun Point.plus(other: Point): Point {
3         return Point(x + other.x, y + other.y)
4     }
5
6     >>> val p1 = Point(10, 20)
7     >>> val p2 = Point(30, 40)
8     >>> println(p1 + p2)
9     Point(x=40, y=60)
```

Unpack arguments with spread operator [edit]

Similar to Python, the spread operator asterisk (*) unpacks an array's contents as comma-separated arguments to a function:

```
fun main(args: Array<String>) {
    val list = listOf("args: ", *args)
    println(list)
}
```