## Exercise 1: Basic Function

**Task**: Write a function that returns the phrase "Hello Kotlin".
**Expected Output**:

```
Hello Kotlin
```

**Answer**:

```kotlin
fun greet(): String {
    return "Hello Kotlin"
}

println(greet())
```

**Explanation**: This exercise introduces basic function syntax. Common mistakes include incorrect function naming or forgetting the return type.

---

## Exercise 2: Sum Function

**Task**: Create a function that takes two integers and returns their sum.
**Expected Output** (for inputs 5 and 7):

```
12
```

**Answer**:

```kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}

println(sum(5, 7))
```

**Explanation**: This teaches basic arithmetic operations and function parameters. A common issue is mismatching parameter types or missing return statements.

---

## Exercise 3: String Interpolation

**Task**: Write a function that takes a name and returns a greeting message using string interpolation.

**Expected Output** (for input "Kotlin"):

```
Hello, Kotlin!
```

**Answer**:

```kotlin
fun greeting(name: String): String {
    return "Hello, $name!"
}

println(greeting("Kotlin"))
```

**Explanation**: Introduces string interpolation. Common mistakes involve incorrect syntax for string interpolation or concatenation.

---

## Exercise 4: Conditional Statements

**Task**: Write a function that checks if a number is even and returns a corresponding boolean value.

**Expected Output** (for input 4):

```
true
```

**Answer**:

```kotlin
fun isEven(number: Int): Boolean {
    return number % 2 == 0
}

println(isEven(4))
```

**Explanation**: Covers the modulus operator and boolean return type. A typical issue is incorrect implementation of the even check.

---

## Exercise 5: When Expression

**Task**: Use `when` to classify an integer as "positive", "negative", or "zero".
**Expected Output** (for input -5):

```
negative
```

**Answer**:

```
fun classifyNumber(number: Int): String {
    return when {
        number > 0 -> "positive"
        number < 0 -> "negative"
        else -> "zero"
    }
}

println(classifyNumber(-5))
```

**Explanation**: Teaches the use of `when` for multiple conditions. Common pitfalls include missing the `else` case or overlapping conditions.

---

## Exercise 6: Lists and Loops

**Task**: Create a function that takes a list of integers and returns the sum of all elements.
**Expected Output** (for input `[1, 2, 3, 4, 5]`):

```
15
```

**Answer**:

```
fun sumOfList(numbers: List<Int>): Int {
    var sum = 0
    for (number in numbers) {
        sum += number
    }
    return sum
}

println(sumOfList(listOf(1, 2, 3, 4, 5)))
```

**Explanation**: Introduces loops and list handling. Common issues include incorrect loop syntax or variable initialization.

---

## Exercise 7: Data Classes

**Task**: Create a data class `Person` with two properties: `name` (String) and `age` (Int). Then, create an instance of `Person` and print it.
**Expected Output**:

```
Person(name=John, age=30)
```

**Answer**:

```
data class Person(val name: String, val age: Int)

val john = Person("John", 30)
println(john)
```

**Explanation**: Teaches data class creation and instantiation. Common pitfalls include misunderstanding the automatic generation of `toString`, `equals`, and `hashCode` methods.

---

## Exercise 8: Default and Named Arguments

**Task**: Write a function with default arguments that returns a greeting message. Use named arguments to call this function.
**Expected Output**:

```
Hello, Kotlin Developer!
```

**Answer**:

```
fun greetUser(name: String = "Kotlin", role: String = "Developer"):
String {
    return "Hello, $name $role!"
}

println(greetUser(name = "Kotlin"))
```

**Explanation**: Introduces default values for function parameters and the use of named arguments. Common mistakes include incorrect default value usage or syntax errors in named arguments.

---

## Exercise 9: Null Safety

**Task**: Create a function that takes a nullable String and returns its length or 0 if it is null.
**Expected Output** (for input `"Kotlin"`):

```
6
```

**Answer**:

```
fun stringLength(str: String?): Int {
    return str?.length ?: 0
}

println(stringLength("Kotlin"))
```

**Explanation**: Covers null safety, safe calls, and the Elvis operator. Common issues include mishandling null values or misunderstanding the Elvis operator.

---

## Exercise 10: Extension Functions

**Task**: Write an extension function for `String` that reverses the string “FoodPanda” and appends an exclamation mark.
**Expected Output** (for input `"FoodPanda"`):

```
adnaPooF!
```

**Answer**:

```
fun String.reverseAndExclaim(): String {
    return this.reversed() + "!"
}

println("Kotlin".reverseAndExclaim())
```

**Explanation**: Teaches creating extension functions. Common pitfalls include incorrect extension function syntax or misunderstanding the `this` keyword in the context of extension functions.

---

## Exercise 11: Simple Loop

**Task**: Write a function that prints numbers from 1 to 5 using a `for` loop.
**Expected Output**:

```
1 2 3 4 5
```

**Answer**:

```kotlin
fun printNumbers() {
    for (i in 1..5) {
        print("$i ")
    }
}

printNumbers()
```

**Explanation**: This exercise introduces the basic `for` loop and range expression in Kotlin. A common issue is incorrect range usage.

---

## Exercise 12: Basic String Manipulation (Repeated Question)

**Task**: Write a function that takes a string and returns it in reverse order.
**Expected Output** (for input "Kotlin"):

```
niltoK
```

**Answer**:

```kotlin
fun reverseString(str: String): String {
    return str.reversed()
}

println(reverseString("Kotlin"))
```

**Explanation**: This exercise covers basic string functions. Common pitfalls include manually reversing a string instead of using the built-in function.

---

## Exercise 13: Using Lists

**Task**: Create a function that takes a list of numbers and returns the largest number.
**Expected Output** (for input `[1, 3, 2]`):

```
3
```

**Answer**:

```
fun findMax(numbers: List<Int>): Int {
    return numbers.maxOrNull() ?: throw IllegalArgumentException("List
is empty")
}

println(findMax(listOf(1, 3, 2)))
```

**Explanation**: Teaches how to use list functions and handle empty lists. Common issues include not handling the case of an empty list.

---

## Exercise 14: Basic Class

**Task**: Define a simple class `Car` with properties `make` and `year`. Create an instance of this class and print its properties.
**Expected Output**:

```
Make: Toyota, Year: 2020
```

**Answer**:

```
class Car(val make: String, val year: Int)

fun main() {
    val myCar = Car("Toyota", 2020)
    println("Make: ${myCar.make}, Year: ${myCar.year}")
}
```

**Explanation**: Introduces class definition and instantiation. Common pitfalls include incorrect syntax for defining properties or misunderstanding class constructors.

---

## Exercise 15: Using `if` Expression

**Task**: Write a function that takes an integer and returns "Odd" if the number is odd and "Even" if it's even.
**Expected Output** (for input 3):

```
Odd
```

**Answer**:

```
fun checkOddEven(number: Int): String {
    return if (number % 2 == 0) "Even" else "Odd"
}

println(checkOddEven(3))
```

**Explanation**: Covers the `if` expression and modulus operator. Common issues include incorrect implementation of the even-odd logic.

---

## Exercise 16: Default Parameters

**Task**: Create a function with a default parameter that prints a greeting message. The name should be a parameter with a default value.
**Expected Output** (without passing a parameter):

```
Hello, John!
```

**Answer**:

```
fun greet(name: String = "John") {
    println("Hello, $name!")
}

greet()
```

**Explanation**: Teaches the use of default parameters. A common mistake is not providing a default value for the parameter.

---

## Exercise 17: Simple `when` Statement

**Task**: Write a function using `when` that takes an integer and returns "small" if it's less than 5, "big" if it's greater than or equal to 5.
**Expected Output** (for input 7):

```
big
```

**Answer**:

```
fun sizeDescription(number: Int): String {
    return when {
        number < 5 -> "small"
        else -> "big"
    }
}

println(sizeDescription(7))
```

**Explanation**: Introduces `when` statement. Common pitfalls include not covering all possible cases or incorrect condition logic.

---

## Exercise 18: Concatenating Lists

**Task**: Write a function that concatenates two lists of integers and returns the result.
**Expected Output** (for inputs `[1, 2]` and `[3, 4]`):

```
[1, 2, 3, 4]
```

**Answer**:

```
fun concatenateLists(list1: List<Int>, list2: List<Int>): List<Int> {
    return list1 + list2
}

println(concatenateLists(listOf(1, 2), listOf(3, 4)))
```

**Explanation**: Covers basic list operations. A common issue is incorrectly merging lists or creating unnecessary complexity in concatenation.

---

## Exercise 19: Counting Characters

**Task**: Create a function that counts the number of times a specific character appears in a string.
**Expected Output** (for inputs `"hello"` and `'l'`):

```
2
```

**Answer**:

```
fun countChar(str: String, charToCount: Char): Int {
    return str.count { it == charToCount }
}

println(countChar("hello", 'l'))
```

**Explanation**: Introduces the `count` function and lambda expressions. Common mistakes include manual iteration over the string or incorrect lambda usage.

---

## Exercise 20: Simple Null Check

**Task**: Write a function that takes a nullable integer and returns "Yes" if it's not null and "No" if it is null.

**Expected Output** (for a null input):

```
No
```

**Answer**:

```
fun isNotNull(input: Int?): String {
    return if (input != null) "Yes" else "No"
}

println(isNotNull(null))
```

**Explanation**: Teaches basic nullability checks. Common issues include misunderstanding nullable types or incorrect implementation of null checks