Practice Problems: Kotlin Basics

1. Before you begin

While you've put in the hard work to gain knowledge of the basics of Kotlin programming, now it's time to put what you learned into practice.

These exercises test your understanding of the concepts that you studied. They're themed around real-world use cases, some of which you probably encountered before.

Follow the instructions to find a solution for each problem. If you get stuck, some of the exercises have hints that can help you. The solution code is available at the end, but it's strongly recommended that you solve the exercises before you check your answers.

Consider the solutions as one way to solve the problems and feel free to experiment however you feel comfortable. You can solve some of the exercises in multiple ways, and use different names for the functions and variables.

Work through the problems at a pace that's comfortable to you. There is a duration listed, but it is not necessary to adhere to those timings, as they are only estimates. You are encouraged to take as much time as you need to solve each problem thoughtfully.

It's recommended that you use Kotlin Playground (https://developer.android.com/training/kotlinplayground) to solve these exercises.

Prerequisites

- Familiarity with Kotlin Playground
- Ability to define and call functions
- Knowledge of basic Kotlin data types
- Knowledge of immutable and mutable variables
- Knowledge of the `println()` function
- Completion of the Your first program in Kotlin, Create and use variables in Kotlin, and Create and use functions in Kotlin codelabs

What you'll need

- A computer with internet access and a web browser

2. Print messages

Tell your friends what you learned in this pathway.

- Can you write a `main()` function that prints these messages on four separate lines?

```
Use the val keyword when the value doesn't change.
Use the var keyword when the value can change.
When you define a function, you define the parameters that can be passed to it.
When you call a function, you pass arguments for the parameters.
```

3. Fix compile error

This program prints a message that notifies the user that they received a chat message from a friend.

```
fun main() {
    println("New chat message from a friend'}
}
```

1. Can you figure out the root cause of the compile errors in this program and fix them?
2. Does the code use appropriate symbols to indicate the open and close of the string and function argument?

Hint: You can use Kotlin Playground to run the code and view the compilation errors.

After you fix the errors, the program should compile without errors and print this output:

```
New chat message from a friend
```

4. String templates

This program informs users about the upcoming promotional sale on a particular item. It has a string template, which relies on the `discountPercentage` variable for the percent discount and the `item` variable for the item on sale. However, there are compilation errors in the code.

```
fun main() {
    val discountPercentage: Int = 0
```

```
    val offer: String = ""
    val item = "Google Chromecast"
    discountPercentage = 20
    offer = "Sale - Up to $discountPercentage% discount on
$item! Hurry up!"

    println(offer)
}
```

1. Can you figure out the root cause of the errors and fix them?
2. Can you determine the output of this program before you run the code in Kotlin Playground?

Hint: Can you re-assign a value to a read-only variable?

After you fix the errors, the program should compile without errors and print this output:

```
Sale - Up to 20% discount on Google Chromecast! Hurry up!
```

5. String concatenation

This program displays the total party size. There are adults and kids at the party. The `numberOfAdults` variable holds the number of adults at the party and the `numberOfKids` variable holds the number of kids.

```
fun main() {
    val numberOfAdults = "20"
    val numberOfKids = "30"
    val total = numberOfAdults + numberOfKids
    println("The total party size is: $total")
}
```

Step 1

- Can you determine the output of this program before you run the code in Kotlin Playground?

After you determine the output, run the code in Kotlin Playground and then check if your output matches the output displayed.

Hint: What happens when you use the + operator on two strings?

Step 2

The code works and prints some output, but the output doesn't show the total number of people attending the party.

- Can you find the issue in the code and fix it so that it prints this output?

```
The total party size is: 50
```

6. Message formatting

This program displays the total salary that an employee receives this month. The total salary is divided in two parts: the `baseSalary` variable, which the employee receives every month, and the `bonusAmount` variable, which is an additional bonus awarded to the employee.

```
fun main() {
    val baseSalary = 5000
    val bonusAmount = 1000
    val totalSalary = "$baseSalary + $bonusAmount"
    println("Congratulations for your bonus! You will receive
a total of $totalSalary (additional bonus).")
}
```

1. Can you figure out the output of this code before you run it in Kotlin Playground?
2. When you run the code in Kotlin Playground, does it print the output that you expected?

7. Implement basic math operations

In this exercise, you write a program that performs basic math operations and prints the output.

Step 1

This `main()` function contains one compile error:

```
fun main() {
    val firstNumber = 10
    val secondNumber = 5
```

```
    println("$firstNumber + $secondNumber = $result")
}
```

- Can you fix the error so that the program prints this output?

```
10 + 5 = 15
```

Step 2

The code works, but the logic for adding two numbers is located within the result variable, making your code less flexible to reuse. Instead, you can extract the addition operation into an `add()` function so that the code is reusable. To do this, update your code with the code listed below. Notice that the code now introduces a new `val` called `thirdNumber` and prints the result of this new variable with `firstNumber`.

```
fun main() {
    val firstNumber = 10
    val secondNumber = 5
    val thirdNumber = 8

    val result = add(firstNumber, secondNumber)
    val anotherResult = add(firstNumber, thirdNumber)

    println("$firstNumber + $secondNumber = $result")
    println("$firstNumber + $thirdNumber = $anotherResult")
}

// Define add() function below this line
```

- Can you define the add() function so that the program prints this output?

```
10 + 5 = 15
10 + 8 = 18
```

Step 3

Now you have a reusable function to add two numbers.

- Can you implement the `subtract()` function the same way you implemented the `add()` function? Modify the `main()` function as well to use the `subtract()` function so you can verify that it works as expected.

Hint: Think about the difference between addition, subtraction and other math operations. Start work on the solution code from there.

8. Default parameters

Gmail has a feature that sends a notification to the user whenever a login attempt is made on a new device.

In this exercise, you write a program that displays a message to users with this message template:

```
There's a new sign-in request on operatingSystem for your
Google Account emailId.
```

You need to implement a function that accepts an `operatingSystem` parameter and an `emailId` parameter, constructs a message in the given format, and returns the message.

For example, if the function was called with "`Chrome OS`" for the `operatingSystem` and "`sample@gmail.com`" for the `emailId`, it should return this string:

There's a new sign-in request on Chrome OS for your Google Account sample@gmail.com.

Step 1

1. Can you implement the `displayAlertMessage()` function in this program so that it prints the output displayed?

```
fun main() {
    val operatingSystem = "Chrome OS"
    val emailId = "sample@gmail.com"

    println(displayAlertMessage(operatingSystem, emailId))
}

// Define your displayAlertMessage() below this line.
```

2. Does your program print this output?

```
There's a new sign-in request on Chrome OS for your Google
Account sample@gmail.com.
```

Step 2

Great job! You displayed the message. However, in some cases, you discover that you can't determine the user's operating system. In such cases, you need to specify the operating system name as `Unknown OS`. You can further optimize the code so that you don't need to pass the `Unknown OS` argument each time that the function is called.

1. Can you find a way to optimize the code with this information so that it prints this output?

```
There's a new sign-in request on Unknown OS for your Google
Account user_one@gmail.com.

There's a new sign-in request on Windows for your Google
Account user_two@gmail.com.

There's a new sign-in request on Mac OS for your Google
Account user_three@gmail.com.
```

2. To print the above message, replace the `main()` function implementation with this one:

```
fun main() {
    val firstUserEmailId = "user_one@gmail.com"

    // The following line of code assumes that you named your
parameter as emailId.
    // If you named it differently, feel free to update the
name.
    println(displayAlertMessage(emailId = firstUserEmailId))
    println()

    val secondUserOperatingSystem = "Windows"
    val secondUserEmailId = "user_two@gmail.com"

    println(displayAlertMessage(secondUserOperatingSystem,
secondUserEmailId))
    println()

    val thirdUserOperatingSystem = "Mac OS"
    val thirdUserEmailId = "user_three@gmail.com"

    println(displayAlertMessage(thirdUserOperatingSystem,
thirdUserEmailId))
    println()
}
```

9. Pedometer

The pedometer is an electronic device that counts the number of steps taken. Nowadays, almost all mobile phones, smart watches, and fitness gear come with pedometers built into them. The health and fitness app uses built-in pedometers to calculate the number of steps taken. This function calculates the number of calories that the user burns based on the user's number of steps.

- Can you rename the functions, function parameters, and variables in this program based on best practices?

```
fun main() {
    val Steps = 4000
    val caloriesBurned = PEDOMETERstepsTOcalories(Steps);
    println("Walking $Steps steps burns $caloriesBurned
calories")
}

fun PEDOMETERstepsTOcalories(NumberOFStepS: Int): Double {
    val CaloriesBURNEDforEachStep = 0.04
    val TotalCALORIESburned = NumberOFStepS *
CaloriesBURNEDforEachStep
    return TotalCALORIESburned
}
```

10. Compare two numbers

Modern mobile phones have a built-in feature that tracks screen time, or the time you spend on your phone each day.

In this exercise, you implement a function that compares the time in minutes that you spent on your phone today versus the time spent yesterday. The function accepts two integer parameters and returns a boolean value.

The first parameter holds the number of minutes that you spent today and the second parameter holds the number of minutes that you spent yesterday. The function returns a `true` value if you spent more time on the phone today compared to yesterday. Otherwise, it returns a `false` value.

For example, if you called the function with these named arguments:

- `timeSpentToday = 300` and `timeSpentYesterday = 250`, the function returns a `true` value.
- `timeSpentToday = 300` and `timeSpentYesterday = 300`, the function returns a `false` value.

- `timeSpentToday = 200` and `timeSpentYesterday = 220`, the function returns a `false` value.

Hint: The `>` comparison operator returns a `true` value if the value before the operator is greater than the value after it. Otherwise, it returns a `false` value.

11. Move duplicate code into a function

This program displays the weather for different cities. It includes the city name, the high and low temperature for the day, and the chance of rain.

```
fun main() {
    println("City: Ankara")
    println("Low temperature: 27, High temperature: 31")
    println("Chance of rain: 82%")
    println()

    println("City: Tokyo")
    println("Low temperature: 32, High temperature: 36")
    println("Chance of rain: 10%")
    println()

    println("City: Cape Town")
    println("Low temperature: 59, High temperature: 64")
    println("Chance of rain: 2%")
    println()

    println("City: Guatemala City")
    println("Low temperature: 50, High temperature: 55")
    println("Chance of rain: 7%")
    println()
}
```

There are many similarities in the code that prints the weather for each city. For example, there are phrases that are repeated multiple times, such as `"City:"` and `"Low temperature:"`. Similar, repeated code creates the risk of errors in your program. For one of the cities, you may have a typo or you may forget one of the weather details.

1. Can you create a function that prints the weather details for a single city to reduce the repetition in the `main()` function and then do the same for the remaining cities?
2. Can you update the `main()` function to call the function that you created for each city and pass in the appropriate weather details as arguments?

12. Solution code

Print messages

The solution uses the `println()` function to print the messages on each line.

```
fun main() {
    println("Use the val keyword when the value doesn't
change.")
    println("Use the var keyword when the value can change.")
    println("When you define a function, you define the
parameters that can be passed to it.")
    println("When you call a function, you pass arguments for
the parameters.")
}
```

Fix compile error

The code contained two compilation errors:

1. The string should end with a double quotation mark rather than a single quotation mark.
2. The function argument should end with a parenthesis rather than a curly bracket.

```
fun main() {
    println("New chat message from a friend")
}
```

String templates

The compilation errors are the result from the assignment of the `discountPercentage` and `offer` read-only variables to new values; this assignment isn't allowed.

```
fun main() {
    val discountPercentage = 20
    val item = "Google Chromecast"
    val offer = "Sale  - Up to $discountPercentage% discount
off $item! Hurry Up!"

    println(offer)
}
```

As an alternative solution, you could declare the `discountPercentage` integer and `offer` string with the `var` keyword. However, their values are immutable in the context of the program, so you can stick with the `val` keyword.

String concatenation

Step 1

The program prints this output:

```
The total party size is: 2030
```

This was a trick question. When the `+` operator is used on `String` values, it produces a concatenated string. The integers are wrapped in double quotation marks, so they're treated as strings instead of integers, hence the output of `2030`.

Step 2

You could remove the double quotation marks around the `numberOfAdults` and `numberOfKids` variables to convert them to Int variables.

```
fun main() {
    val numberOfAdults = 20
    val numberOfKids = 30
    val total = numberOfAdults + numberOfKids
    println("The total party size is: $total")
}
```

If you remember, the Kotlin compiler can infer the type of variables based on the values assigned to them. In this case, the compiler infers that the `numberOfAdults` and `numberOfKids` variables are `Int` types.

Message formatting

The program prints this output:

```
Congratulations for your bonus! You will receive a total of
5000 + 1000 (additional bonus).
```

`"$baseSalary + $bonusAmount"` uses the template expressions syntax. In template expressions, the code is evaluated first and then the result is concatenated in a string.

In the question, the `$baseSalary` variable is evaluated to a `5000` value and the `$bonusAmount` variable is evaluated to a `1000` value. Then, the result is concatenated to produce `"5000 + 1000"` and assigned to the `result` variable.

Implement basic math operations

Step 1

Define an immutable `result` variable with the `val` keyword and then assign the result of the addition operation to it:

```
fun main() {
    val firstNumber = 10
    val secondNumber = 5

    val result = firstNumber + secondNumber
    println("$firstNumber + $secondNumber = $result")
}
```

Step 2

1. Create an `add()` function that accepts a `firstNumber` parameter and `secondNumber` parameter, both of `Int` type, and returns an Int value.
2. Enter the code for the addition operation inside the `add()` function body and then use the `return` keyword to return the result of the operation.

```
fun add(firstNumber: Int, secondNumber: Int): Int {
    return firstNumber + secondNumber
}
```

Step 3

1. Define a `subtract()` function that accepts a `firstNumber` parameter and `secondNumber` parameter, both of `Int` type, and returns an `Int` value.
2. Enter the code for the subtraction operation inside the `subtract()` function body and then use the `return` keyword to return the result of the operation.

```
fun subtract(firstNumber: Int, secondNumber: Int): Int {
    return firstNumber - secondNumber
}
```

3. Modify the `main()` function to use the new `subtract()` function. An example solution could look like this:

```
fun main() {
    val firstNumber = 10
    val secondNumber = 5
    val thirdNumber = 8

    val result = add(firstNumber, secondNumber)
    val anotherResult = subtract(firstNumber, thirdNumber)
```

```
    println("$firstNumber + $secondNumber = $result")
    println("$firstNumber - $thirdNumber = $anotherResult")
}

fun add(firstNumber: Int, secondNumber: Int): Int {
    return firstNumber + secondNumber
}

fun subtract(firstNumber: Int, secondNumber: Int): Int {
    return firstNumber - secondNumber
}
```

Default parameters

Step 1

1. Create a `displayAlertMessage()` function that accepts an `operatingSystem` parameter and `emailId` parameter, both of `String` type, and returns a `String` value.
2. In the function body, use a template expression to update the message and return it.

```
fun displayAlertMessage(operatingSystem: String, emailId:
String): String {
    return "There is a new sign-in request on $operatingSystem
for your Google Account $emailId."
}
```

Step 2

- Assign an "`Unknown OS`" value to the `operatingSystem` parameter.

```
fun displayAlertMessage(
    operatingSystem: String = "Unknown OS",
    emailId: String
): String {
    return "There is a new sign-in request on $operatingSystem
for your Google Account $emailId."
}
```

Pedometer

Function names and variable names should follow the *camel case* convention.

If the names contain multiple words, lowercase the first letter of the first word, capitalize the first letter of subsequent words, and remove any spaces between the words.

Example function names include:

- `calculateTip`
- `displayMessage`
- `takePhoto`

Example variable names include:

- `numberOfEmails`
- `cityName`
- `bookPublicationDate`

To learn more about names, see Naming rules (https://kotlinlang.org/docs/coding-conventions.html#naming-rules).

Avoid using a Kotlin keyword (https://kotlinlang.org/docs/keyword-reference.html) as a function name because those words are already assigned specific meanings in the Kotlin language.

Your solution code should look something like this code snippet:

```
fun main() {
    val steps = 4000
    val caloriesBurned = pedometerStepsToCalories(steps)
    println("Walking $steps steps burns $caloriesBurned
calories")
}

fun pedometerStepsToCalories(numberOfSteps: Int): Double {
    val caloriesBurnedForEachStep = 0.04
    val totalCaloriesBurned = numberOfSteps *
caloriesBurnedForEachStep
    return totalCaloriesBurned
}
```

Compare two numbers

- Create a `compareTime()` function that accepts a `timeSpentToday` parameter and a `timeSpentYesterday` parameter, both of `Int` type, and returns a `Boolean` value.

The solution relies on the `>` comparison operator. The operator evaluates to a `Boolean` value, so the `compareTime()` function simply returns the result of `timeSpentToday > timeSpentYesterday`.

For example, if you pass a `300` argument to the `timeSpentToday` parameter and a `250` argument to the `timeSpentYesterday` parameter, the function body evaluates to `300 > 250`, which returns a `true` value because 300 is greater than 250.

```
fun main() {
    println("Have I spent more time using my phone today:
${compareTime(300, 250)}")
    println("Have I spent more time using my phone today:
${compareTime(300, 300)}")
    println("Have I spent more time using my phone today:
${compareTime(200, 220)}")
}

fun compareTime(timeSpentToday: Int, timeSpentYesterday: Int):
Boolean {
    return timeSpentToday > timeSpentYesterday
}
```

```
Have I spent more time using my phone today: true
Have I spent more time using my phone today: false
Have I spent more time using my phone today: false
```

Move duplicate code into a function

1. Create a function that prints out the weather details for the city of Ankara after the `main()` function.

For the function name, you can use `printWeatherForCity()` or something similar.

2. Call the function from the `main()` function.

The program should print the weather details for Ankara.

```
fun main() {
    printWeatherForCity()
}

fun printWeatherForCity() {
    println("City: Ankara")
    println("Low temperature: 27, High temperature: 31")
    println("Chance of rain: 82%")
```

```
    println()
}
```

Now you can create another function that's more flexible so that it can print weather details for other cities.

3. Replace the Ankara-specific parts of the `println()` statements with variables.

Remember to use camel case convention for the variable names and the `$` symbol before the variable so that the value of the variable gets used instead of the variable name. These are string templates which you learned about in an earlier codelab.

```
fun printWeatherForCity() {
    println("City: $cityName")
    println("Low temperature: $lowTemp, High temperature:
$highTemp")
    println("Chance of rain: $chanceOfRain%")
    println()
}
```

4. Change the function definition so that those variables are parameters that must be passed into the function when it's called and specify the data type for each parameter.

The `cityName` parameter is of `String` type, while the `lowTemp`, `highTemp`, and `chanceOfRain` parameters are of Int type.

There's no `return` value needed in the function definition because the messages are printed to the output.

```
fun printWeatherForCity(cityName: String, lowTemp: Int,
highTemp: Int, chanceOfRain: Int) {
    println("City: $cityName")
    println("Low temperature: $lowTemp, High temperature:
$highTemp")
    println("Chance of rain: $chanceOfRain%")
    println()
}
```

5. Update the `main()` function to call the `printWeatherForCity()` function and pass in the weather details for Ankara.

The city name is `"Ankara"`, the low temperature is `27`, the high temperature is `31`, and the chance of rain is `82`.

```
fun main() {
    printWeatherForCity("Ankara", 27, 31, 82)
```

```
}

fun printWeatherForCity(cityName: String, lowTemp: Int,
highTemp: Int, chanceOfRain: Int) {
    println("City: $cityName")
    println("Low temperature: $lowTemp, High temperature:
$highTemp")
    println("Chance of rain: $chanceOfRain%")
    println()
}
```

6. Run the program to verify that the output shows the weather details for Ankara.
7. Call the `printWeatherForCity()` function with the weather details for the other cities.

```
fun main() {
    printWeatherForCity("Ankara", 27, 31, 82)
    printWeatherForCity("Tokyo", 32, 36, 10)
    printWeatherForCity("Cape Town", 59, 64, 2)
    printWeatherForCity("Guatemala City", 50, 55, 7)
}

fun printWeatherForCity(cityName: String, lowTemp: Int,
highTemp: Int, chanceOfRain: Int) {
    println("City: $cityName")
    println("Low temperature: $lowTemp, High temperature:
$highTemp")
    println("Chance of rain: $chanceOfRain%")
    println()
}
```

8. Run the program.

It should print the same output as the original program, but now your code is more concise and doesn't contain unnecessary repetition! All the code for printing weather details of a city is centralized in a single place: the `printWeatherForCity()` function. If you ever want to change how weather details are displayed, you can change them in a single place that applies to all the cities.

13. Additional Practice

For more practice on the Kotlin language, check out the Kotlin Core track by JetBrains Academy (https://hyperskill.org/tracks/18). To jump to a specific topic, go to the knowledge map (https://hyperskill.org/knowledge-map) to view the list of topics covered in the track.