Session 1: No AI Assistant

Rules:

1. You will receive 3 problems. Solve them in the order shown (top to bottom).
2. You can use the regular sources: official documentation, stackoverflow, google search etc.
3. **Do not use any AI assistant!**
4. Write your code in an Android or iOS application environment.
   - See documents in root for IDE installation if needed.
   - You will need to upload the project in the root of your folder.
5. Each problem has a time limit. If you exceed the time limit please stop and move to the next task (if any remaining).
6. Time your session. As soon as you think the solution is complete. Stop the timer and add it into the function documentation.
7. Screen record your session. You can record only the desktop/screen with the IDE.
   - You will upload your recording to the folder of this file.
8. Please do not scroll to the problems section (below) until you are prepared to start (editor ready, recording ready).
9. Upload the resulted project in the root of your folder.
10. Maximum time for entire session: 1h.

Example:

Task:
Write a program that takes two numbers as input and determines the maximum of the two numbers. The program should output the maximum number.

Solution:

```
// Time: 3:45
fun max(a: Int, b: Int): Int {
   return if (a > b) a else b
}
```

==

Tasks below… stop scrolling if you are not ready…

==

Task 1: SL3
Time limit: 10 minutes

Write an extension function to the platform-specific locale type that returns true if the system has the current language set to Romanian.

==

==
Task 2: SM1
Time limit: 20 minutes

Write a platform-specific asynchronous function that receives an array of integers and returns the minimum and maximum values in the array using a tuple (pair) type after a delay of one second.

==

==
Task 3:
Time limit: 30 minutes

Write an abstract type `ErrorTranslatable` with two methods:

- `userFacingReason` receiving a platform-specific error and returning a String to be shown to the user, specific to the error if the error is recognizable if not a generic message.
- `userFacingCode` receives a platform-specific error and returns an optional Int to be shown to the user if the error is cognizable.

Write a concrete type that implements the abstract type. Provide default behavior for the two methods if not overridden by a concrete type. Use type extensions to implement the behavior of the abstract type.

==