Things we are very picky about:

Test Coverage: The solution should be developed "test-first", and should have good unit test coverage.
Simplicity: We value simplicity as an architectural virtue and a development practice. Solutions should reflect the difficulty of the assigned task, and should not be overly complex. Layers of abstraction, patterns, or architectural features that aren't called for should not be included.
Self-explanatory code: The solution you produce must speak for itself. Multiple paragraphs explaining the solution are a sign that it isn't straightforward enough to understand purely by reading code, and are not appropriate.

Exercise:

Imagine you're creating a footer with pagination to browse through the several pages of a given website.

Let's assume the usage of the following variables for the effect:
- **current_page**
- **total_pages**
- **boundaries**: how many pages we want to link in the beginning, or end (meaning, how many pages starting at page 1 and how many leading up to the last page, inclusive)
- **around**: how many pages we want to link before and after the current page, exclusive.
For pages with no direct link we should use one set of three points (...) per set of pages hidden.

Some examples:
1) **current_page** = 4; **total_pages** = 5; **boundaries** = 1; **around** = 0
Expected result: 1 ... 4 5

2) **current_page** = 4; **total_pages** = 10; **boundaries** = 2; **around** = 2
Expected result: 1 2 3 4 5 6 ... 9 10

The code should print the result and not only calculate it.

Hints:

- Focus on what we're asking, don't overdue or get lost.
- Make sure it runs efficiently regardless of the total pages' number.
- Keep it simple, if you're writing a lot of code, you're most likely overcomplicating.
- Keep it readable, we like to be able to understand quickly what you wrote.
- Test the solution yourself first, thinking about corner-cases. We're going to extensively test your code.