PA 5 Task Card – Decode the Scrambled Message

Today you will use the `stringr` package to work with a character vector to decode a message.

This task is complex. It requires many different types of abilities. Everyone will be good at some of these abilities but nobody will be good at all of them. In order to solve this puzzle, you will need to use the skills of each member of your group.

Group Roles

When you are the **Developer**, you will type into the Quarto document in RStudio. However, you **do not** type your own ideas. Instead, you type what the Coder tells you to type. You are permitted to ask the Coder clarifying questions, and, if both of you have a question, you are permitted to ask the professor. You are expected to run the code provided by the Coder and, if necessary, to work with the Coder to debug the code. Once the code runs, you are expected to collaborate with the Coder to write code comments that describe the actions taken by your code.

When you are the **Coder**, you are responsible for reading the instructions / prompts and directing the Developer what to type in the Quarto document. You are responsible for managing the resources your group has available to you (e.g., cheatsheet, textbook). If necessary, you should work with the Developer to debug the code you specified. Once the code runs, you are expected to collaborate with the Developer to write code comments that describe the actions taken by your code.

Group Norms

Remember, your group is expected to adhere to the following norms:

1. Think and work together. Do not divide the work.
2. You are smarter together.
3. Be open minded.
4. No cross-talk with other groups.
5. Communicate with each other!

`stringr`

Task	stringr
Replace `pattern` with `replacement`	`str_replace(x, pattern, replacement)` and `str_replace_all(x, pattern, replacement)`
Convert case	`str_to_lower(x)`, `str_to_upper(x)`, `str_to_title(x)`
Find the length of a string	`str_length()`
Strip whitespace from start/end	`str_trim(x)` , `str_squish(x)`
Pad strings to a specific length	`str_pad(x, …)`
Test if the string contains a pattern	`str_detect(x, pattern)`
Count how many times a pattern appears in the string	`str_count(x, pattern)`
Find the first appearance of the pattern within the string	`str_locate(x, pattern)`
Find all appearances of the pattern within the string	`str_locate_all(x, pattern)`
Detect a match at the start/end of the string	`str_starts(x, pattern)` ,`str_ends(x, pattern)`
Subset a string from index a to b	`str_sub(x, a, b)`
Convert string encoding	`str_conv(x, encoding)`

Regex – As input into `stringr` functions

Task	Regular Expression
Find a set of characters	`"[abc]"` — matches any single character `a`, `b`, or `c` `"[A-z]"` — matches all upper and lower case letters
Match **any** character	`"\\."`
Locate punctuation	`"[:punct:]"`
Locate words	`"\\w"`
Locate anything alphanumeric	`"[:alnum:]"`
Starts with a z	`"^z"`
Ends with a w	`"$w"`
Does not contain a z	`[^z]`
Find Repeated Patterns - Vague	• * means repeat between 0 and inf times • + means 1 or more times • `?` means 0 or 1 times – most useful when you're looking for something optional
Find repeated patterns - Specific	• `{a}` means repeat exactly `a` times • `{a, b}` means repeat between `a` and `b` times