**Introduction to the PEDAC Process**

* P – [Understand the] Problem
* E – Examples / Test cases
* D – Data Structure
* A – Algorithm
* C – Code

P – [Understand the] Problem

1. Read the problem description
2. Check the test cases, if any
3. If any part of the problem is unclear, ask the interviewer or problem requester to clarify the matter

**# PROBLEM:**

# Given a string, write a method change\_me which returns the same

# string but with all the words in it that are palindromes uppercased.

# change\_me("We will meet at noon") == "We will meet at NOON"

# change\_me("No palindromes here") == "No palindromes here"

# change\_me("") == ""

# change\_me("I LOVE my mom and dad equally") == "I LOVE my MOM and DAD equally"

Questions to clarify the stated problem:

1. What is a palindrome?
2. Should the words in the string remain the same if they already use uppercase?
3. How should I deal with empty strings provided as input?
4. Can I assume that all inputs are strings?
5. Should I consider letter case when deciding whether a word is a palindrome?
6. Do I need to return the same string object or an entirely new string?
7. Always verify your assumptions either by looking at the test cases or by asking the interviewer: for example – treat strings as case-sensitive or not,

Make sure to do the following:

1. Write down the inputs and outputs for the problem
2. Describe the rules you must follow
3. Rules should include explicit and implicit requirements in the problem

# input: string

# output: string (not the same object)

# rules:

# Explicit requirements:

# - every palindrome in the string must be converted to

# uppercase. (Reminder: a palindrome is a word that reads

# the same forwards and backward).

# - Palindromes are case sensitive ("Dad" is not a palindrome, but "dad" is.)

# Implicit requirements:

# - the returned string shouldn't be the same string object.

# - if the string is an empty string, the result should be an empty

# string

**Data Structure / Algorithm**

**# PROBLEM:**

# Given a string, write a method `palindrome\_substrings` which returns

# all the substrings from a given string which are palindromes. Consider

# palindrome words case sensitive.

# Test cases:

# palindrome\_substrings("supercalifragilisticexpialidocious") == ["ili"]

# palindrome\_substrings("abcddcbA") == ["bcddcb", "cddc", "dd"]

# palindrome\_substrings("palindrome") == []

# palindrome\_substrings("") == []

P – [Understand the] Problem:

# Some questions you might have?

# 1. What is a substring?

# 2. What is a palindrome?

# 3. Will inputs always be strings?

# 4. What does it mean to treat palindrome words case-sensitively?

# input: string

# output: an array of substrings

# rules:

# Explicit requirements:

# - return only substrings which are palindromes.

# - palindrome words should be case sensitive, meaning "abBA"

# is not a palindrome.

**Incorrect Algorithm:**

1. Initialize a result variable to an empty array
2. Create an array named substring\_arr that contains on substrings of the input string that are at least 2 characters long
3. Loop through the words in the substring\_arr array
4. If the word is a palindrome, append it to the result array
5. Return the result array

This algorithm fails to account for how to find all the substrings for a given string

**Correct Algorithm:**

# - create an empty array called `result` that will contain all required substrings

# - create a `starting\_index` variable (value `0`) for the starting index of a substring

# - start a loop that iterates over `starting\_index` from `0` to the length of the string minus 2

# - create a `num\_chars` variable (value `2`) for the length of a substring

# - start an inner loop that iterates over `num\_chars` from `2` to `string.length - starting\_index`

# - extract a substring of length `num\_chars` from `string` starting at `starting\_index`

# - append the extracted substring to the `result` array

# - increment the `num\_chars` variable by `1`

# - end the inner loop

# - increment the `starting\_index` variable by `1`

# - end the outer loop

# - return the `result` array