Practice Problems for PE05

For **PE05**, you’ll have **two** different problems which you'll need to solve using **recursion**.

1 THE FACTORIAL PROBLEM

Given `n` of 1 or more, return the factorial of `n`, which is `n * (n-1) * (n-2) ... 1`. Compute the result recursively (without loops).

- `factorial(1) → 1`
- `factorial(2) → 2`
- `factorial(3) → 6`

2 THE BUNNY EARS PROBLEM

We have a number of bunnies and each bunny has two big floppy ears. We want to compute the total number of ears across all the bunnies recursively (without loops or multiplication).

- `bunnyEars(0) → 0`
- `bunnyEars(1) → 2`
- `bunnyEars(2) → 4`

3 THE BUNNY EARS II PROBLEM

We have bunnies standing in a line, numbered 1, 2, ... The odd bunnies (1, 3, ..) have the normal 2 ears. The even bunnies (2, 4, ..) we'll say have 3 ears, because they each have a raised foot. Recursively return the number of "ears" in the bunny line 1, 2, ... n (without loops or multiplication).

- `bunnyEars2(0) → 0`
- `bunnyEars2(1) → 2`
- `bunnyEars2(2) → 5`

4 THE TRIANGLE PROBLEM

We have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.

- `triangle(0) → 0`
- `triangle(1) → 1`
- `triangle(2) → 3`

5 THE SUM DIGITS PROBLEM

Given a non-negative `int` n, return the sum of its digits recursively (no loops). Note that mod (%) by `10` yields the rightmost digit (`126 % 10` is 6), while divide (/) by `10` removes the rightmost digit (`126 / 10` is 12).

- `sumDigits(126) → 9`
- `sumDigits(49) → 13`
- `sumDigits(12) → 3`

6 THE COUNT 7 PROBLEM

Given a non-negative int n, return the count of the occurrences of 7 as a digit, so for example 717 yields 2. (no loops). Note that mod (%) by `10` yields the rightmost digit (`126 % 10` is `6`), while divide (/) by `10` removes the rightmost digit (`126 / 10` is 12).

- `count7(717) → 2`
- `count7(7) → 1`
- `count7(123) → 0`

7 THE COUNT 8 PROBLEM

Given a non-negative `int n`, compute recursively (no loops) the count of the occurrences of `8` as a digit, except that an `8` with another `8` immediately to its left counts double, so `8818` yields `4`. Note that mod (%) by `10` yields the rightmost digit (`126 % 10` is `6`), while divide (/) by `10` removes the rightmost digit (`126 / 10` is `12`).

- `count8(8) → 1`
- `count8(818) → 2`

- count8(8818) → 4

8 THE POWER N PROBLEM

Given base and n that are both 1 or more, compute recursively (no loops) the value of base to the n power, so powerN(3, 2) is 9 (3 squared).

- powerN(3, 1) → 3
- powerN(3, 2) → 9
- powerN(3, 3) → 27

9 THE COUNT X PROBLEM

Given a string, compute recursively (no loops) the number of lowercase 'x' chars in the string.

- countX("xxhixx") → 4
- countX("xhixhix") → 3
- countX("hi") → 0

10 THE COUNT HI PROBLEM

Given a string, compute recursively (no loops) the number of times lowercase "hi" appears in the string.

- countHi("xxhixx") → 1
- countHi("xhixhix") → 2
- countHi("hi") → 1

11 THE CHANGE X TO Y PROBLEM

Given a string, compute recursively (no loops) a new string where all the lowercase 'x' chars have been changed to 'y' chars.

- changeXY("codex") → "codey"
- changeXY("xxhixx") → "yyhiyy"
- changeXY("xhixhix") → "yhiyhiy"

12 THE CHANGE PI PROBLEM

Given a string, compute recursively (no loops) a new string where all appearances of "pi" have been replaced by "3.14".

- changePi("xpix") → "x3.14x"
- changePi("pipi") → "3.143.14"
- changePi("pip") → "3.14p"

13 THE NO X PROBLEM

Given a string, compute recursively a new string where all the 'x' chars have been removed.

- `noX("xaxb") → "ab"`
- `noX("abc") → "abc"`
- `noX("xx") → ""`

14 THE ALL STAR PROBLEM

Given a string, compute recursively a new string where all the adjacent chars are now separated by a "*".

- `allStar("hello") → "h*e*l*l*o"`
- `allStar("abc") → "a*b*c"`
- `allStar("ab") → "a*b"`

15 THE PAIR STAR PROBLEM

Given a string, compute recursively a new string where identical chars that are adjacent in the original string are separated from each other by a "*".

- `pairStar("hello") → "hel*lo"`
- `pairStar("xxyy") → "x*xy*y"`
- `pairStar("aaaa") → "a*a*a*a"`

16 THE X TO END PROBLEM

Given a string, compute recursively a new string where all the lowercase 'x' chars have been moved to the end of the string.

- `endX("xxre") → "rexx"`
- `endX("xxhixx") → "hixxxx"`
- `endX("xhixhix") → "hihixxx"`

17 THE COUNT PAIRS PROBLEM

We'll say that a "pair" in a string is two instances of a char separated by a char. So "AxA" the A's make a pair. Pair's can overlap, so "AxAxA" contains 3 pairs -- 2 for A and 1 for x. Recursively compute the number of pairs in the given string.

- `countPairs("axa") → 1`
- `countPairs("axax") → 2`
- `countPairs("axbx") → 1"`

18THE COUNT ABC PROBLEM

Count recursively the total number of "abc" and "aba" substrings that appear in the given string.

- `countAbc("abc") → 1`
- `countAbc("abcxxabc") → 2`
- `countAbc("abaxxaba") → 2`

19THE COUNT 11 PROBLEM

Given a string, compute recursively (no loops) the number of "11" substrings in the string. The "11" substrings should not overlap.

- `count11("11abc11") → 2`
- `count11("abc11x11x11") → 3`
- `count11("111") → 1`

20THE STRING CLEAN PROBLEM

Given a string, return recursively a "cleaned" string where adjacent chars that are the same have been reduced to a single char. So "yyzzza" yields "yza".

- `stringClean("yyzzza") → "yza"`
- `stringClean("abbbcdd") → "abcd"`
- `stringClean("Hello") → "Helo"`

21THE COUNT HI 2 PROBLEM

Given a string, compute recursively the number of times lowercase "hi" appears in the string, however do not count "hi" that have an 'x' immediately before them.

- `countHi2("ahixhi") → 1`
- `countHi2("ahibhi") → 2`
- `countHi2("xhixhi") → 0`

22THE PAREN BIT PROBLEM

Given a string that contains a single pair of parenthesis, compute recursively a new string made of only of the parenthesis and their contents, so "xyz(abc)123" yields "(abc)".

- `parenBit("xyz(abc)123") → "(abc)"`
- `parenBit("x(hello)") → "(hello)"`
- `parenBit("(xy)1") → "(xy)"`

23 THE NEST PAREN PROBLEM

Given a string, return true if it is a nesting of zero or more pairs of parenthesis, like "(())" or "((()))". Suggestion: check the first and last chars, and then recur on what's inside them.

- `nestParen("(())") → true`
- `nestParen("((()))") → true`
- `nestParen("(((x))") → false`

24 THE STR COUNT PROBLEM

Given a string and a non-empty substring sub, compute recursively the number of times that sub appears in the string, without the sub strings overlapping.

- `strCount("catcowcat", "cat") → 2`
- `strCount("catcowcat", "cow") → 1`
- `strCount("catcowcat", "dog") → 0`

25 THE STR COPIES PROBLEM

Given a string and a non-empty substring sub, compute recursively if at least n copies of sub appear in the string somewhere, possibly with overlapping. N will be non-negative.

- `strCopies("catcowcat", "cat", 2) → true`
- `strCopies("catcowcat", "cow", 2) → false`
- `strCopies("catcowcat", "cow", 1) → true`

26 THE STR DIST PROBLEM

Given a string and a non-empty substring sub, compute recursively the largest substring which starts and ends with sub and return its length.

- `strDist("catcowcat", "cat") → 9`
- `strDist("catcowcat", "cow") → 3`
- `strDist("cccatcowcatxx", "cat") → 9`