**PP1 - Regular Polygons 'funsum'**

Consider a regular polygon with the number of sides 'n' and the length of each side 's'.

Area of regular polygon: A quarter of the product of 'n', square of 's' and the reciprocal of tan of angle 'pi' by 'n', where 'pi' is the irrational Mathematical constant.

Perimeter of a polygon: Length of the boundary of the polygon.

Write a function 'funsum' that takes 2 arguments, 'n' and 's'. This function should return the sum of the area and square of the perimeter of the regular polygon, rounded till 4 digits of precision after the decimal point. Paste your Python code for this function in the appropriate space below.

Calculate the value returned by your implemented function 'funsum' for n=7 and s=3. Enter this value in the Result field below.

Example:  
For n=5, s=7, funsum(5, 7) = 1309.3034

**PP2 - "#Sharp" Recursion**

You must have heard about factorial (5!), square root(root 5), etc of a number. Let us assume there exists an operation “#” on integers greater than equal to 2 such that for any integer x >= 2,

x# = ((((((2 ^ 3) ^ 4) ^ 5) …) …) x-1) ^ x)  
Given: 2# = 2

Example:  
3# = (2 ^ 3) = 8  
4# = ((2 ^ 3) ^ 4) = (8 ^ 4) = 4096

1. Write a function “sharp” (using recursion) that takes an integer ‘x’ as argument. This function should return the value of x#. Paste your Python code for this function in the appropriate space below. Write 'NA' if you are unable to design this function.

2. Write a function “ndigits” (using recursion) that takes an integer ‘x’ as argument. This function should return the number of digits in the integer x. Paste your Python code for this function in the appropriate space below. Write 'NA' if you are unable to design this function.

3. Determine the value of the expression:   
Result = ndigits(7#) + 2 \* ndigits(6#) + ndigits(5#) + ndigits(4#)  
You must use the sharp function you designed before for finding the values. Enter the value of the expression in the Result field below. Enter '-1' if you are unable to find this value.

**PP3 - Python Loves Fruits**

Python is an MIT student who loves fruits. He carries different types of fruits (represented by capital alphabets) daily from his house to the MIT campus to eat on the way. But the way he eats fruits is unique. After each fruit he eats (except the last one which he eats just on reaching the campus), he takes a 30 second break in which he buys 1 fruit of each type other than the one he just had. Cobra, his close friend, one day decided to keep a check on Python. He followed him on his way to MIT campus and noted down the type of fruit he ate in the form of a string pattern (Eg.: 'AABBBBCA'). Can you help Cobra determine the maximum quantity out of the different types of fruits that is present with Python when he has reached the campus?

1. Write a function "nfruits" that takes two arguments:

i) A dictionary containing type of fruit and its quantity initially with Python when he leaves home (length < 10)  
ii) A string pattern of the fruits eaten by Python on his journey as observed by Cobra.

This function should return the maximum quantity out of the different types of fruits that is available with Python when he has reached the campus. Paste your Python code for this function in the appropriate space below. Write 'NA' if you are unable to design this function.

2. Result = nfruits({'A': 5, 'B': 7, 'C': 8, 'D': 11, 'E': 4}, 'AAABBBEECAAABBAACCAADDBBAACDCAAABBBAEACCAEADABCB')  
You must use the "nfruits" function you designed before for finding this. Enter the value in the Result field below. Enter '-1' if you are unable to find this value.

Example for clarity:   
Consider the initial quantities as {'A': 1, 'B': 2, 'C': 3} and the string pattern as 'AC'.

'A' is consumed, updated values are {'A': 0, 'B': 2, 'C': 3}  
30 seconds break, Python buys 'B' & 'C', updated values are {'A': 0, 'B': 3, 'C': 4}  
'C' is consumed, updated values are {'A': 0, 'B': 3, 'C': 3}

Now Python has reached the campus. So the function will return 3 that is maximum of the quantities of the three fruits.