CSCI 1133
Exercise Set 7

You should attempt as many of these problems as you can. Practice is *essential* to learning and reinforcing computational problem solving skills. We encourage you to do these without any outside assistance.

Create a directory named `exercise7` and save each of your problem solutions in this directory. You will need to save your source files in this directory and push it to your repository in order to get help/feedback from the TAs. This requires that you follow a rigid set of naming requirements. Name your individual Python source files using "`ex7`" followed by the letter of the exercise, e.g., "`ex7a.py`", "`ex7b.py`", "`ex7c.py`", etc.

Automatic testing of your solutions is performed using a GitHub agent, so it is necessary to name your source files precisely as shown and push them to your repository as you work.

A. **Power Function**

Write a Python module that contains a single pure *recursive* function: `power(a,n)`, that will return the value of a^n given any value of a and some non-negative integer for n. Do not use the Python exponentiation operator (`**`), and do not include any function/method calls other than the recursive call to `power()` :

[Hint: the recurrence involves expressing a^n in terms of a^{n-1}]

B. **Recurrence**

Write a Python module that contains a single pure *recursive* function: `series(n)` that will accept a single integer argument and use *recursion* to compute and return the value of the following series for any positive value $n > 1$:

$$f(n) = 1 + 2 + \frac{3^2}{2} + \frac{4^3}{3} + \frac{5^4}{4} + \ldots + \frac{n^{(n-1)}}{n-1}$$

(For any input values of $n <= 1$, your function should return the value zero)

C. **Counting Digits**

Write a Python module that contains a single pure *recursive* function: `numDigits(n)` that will take any *integer* value as an argument and return the number of digits in the number. Use only basic arithmetic operations. i.e., do not use the built-in function `abs()` and do not use any string or list operations/methods. (Note, your function must work with positive or negative values and must not contain any loop constructs):

D. **Recursive Fun With Strings**

Write a Python module that contains a single pure *recursive* function: `spaceit(istr)`, that takes a single string argument and returns a string in which the adjacent *identical* chars in the original string are separated by a *single* space, i.e.:

```
>>> spaceit('abcdeffghhh')
'abcdef fgh h h'
```

Do not use any string operations other than basic cardinality, concatenation, indexing and slicing, and do not include any function calls other than the recursive call:

E. **Integer to String Conversion**

Write a Python module that contains a single pure *recursive* function: `int2str(n)`, that takes a single integer argument and converts it to a string. e.g:

```
>>> int2str(345)
'345'
```

Do not use any string operations other than concatenation and indexing ("direct lookup"), and do not include any function calls other than the recursive call:

[Hint: use direct look-up on an alphanumeric string to convert the remainder "digit" to its character equivalent]

F. **Radix Conversion**

Write a Python module that contains a single pure *recursive* function: `convertBase(n,base)` that will convert the non-negative decimal number `n` to the specified `base` and return it as a string. The `base` can be any value in the interval [2,16]. Note that decimal values can be converted to a different number base by repeatedly dividing the residual quotient of `n // base` until it is zero, then arranging the remainders in reverse. e.g., to convert 35_{10} to base 2:

```
35 // 2 ==> 17, remainder 1 ('1')
17 // 2 ==>  8, remainder 1 ('1')
 8 // 2 ==>  4, remainder 0 ('0')
 4 // 2 ==>  2, remainder 0 ('0')
 2 // 2 ==>  1, remainder 0 ('0')
 1 // 2 ==>  0, remainder 1 ('1')

35₁₀ == 100011₂
```

Number bases greater than 10 employ successive alphabetic letters to represent digit values greater than 9. For example, 10_{16} is represented using the letter 'A', 11_{16} is 'B' is, 12_{16} is 'C' and so on. e.g., 27_{10} converted to base 16:

```
27 // 16 ==> 1, remainder 11 ('B')
 1 // 16 ==> 0, remainder  1 ('1')

27₁₀ == 1B₁₆
```

[Hint: use direct look-up on an alphanumeric string to convert the remainder "digit" to its character equivalent]

Example:

```
>>> convertBase(47,2)
101111

>>> convertBase(47,16)
2F
```

Constraints:

- Use only basic arithmetic and comparison operations. Do not make any function calls other than the recursive call to `convertBase`.
- Construct the returned string a 'digit' at a time using concatenation.