Python Workshop 6 Exercises

Read and examine the text below and then answer the 8 questions that follow. Note: not all of the answers a solution in code

This workshop is all about recursion. All of the questions (except question 5) require a recursive solution. It does not matter if you are able to answer in some other way, for this workshop, if your solution is not recursive then your answer is WRONG!!

What is recursion?
Recursion occurs when a thing is defined in terms of itself or of its type.
Lists, trees and stacks are recursive data structures. For instance any stack consists of a head item on top of another stack

Figure 1: *A stack plate dispenser*

In mathematics and computer science, a recursion is another type of loop, like the iterative loop (ie. while-loops and for-loops) a recursion allows computer operations to be repeated.

```
def listLen(v): count = 0 for i in v: count += 1 return count
```
A non-recursive routine, ie. a function with an iterative loop.
However, a recursion is a different way of looping and to exhibit recursive behaviour a recursive function can be defined as having 3 major characteristics:

1) A named function with a some data
2) A terminating condition (a base case) that is used to end the recursion
3) The function contains a call to itself where it reduces the complexity of the data towards the base case

Eg
```
def raise(x, n): if n == 0: # base case return 1: else: return (x * raise(x, n-1)))
```
A recursive function to raise x to the power n

How to write a recursive routine

1) Consider the data (or data structure to be processed)
2) Write the base case, the terminating condition for the recursion (and what value it should return)
3) Devise the main calculation or process of the routine
4) Ensure the function calls itself with an operation simplifier its data in some way.

The design of a recursive function

In earlier workbooks we have used different ways to define a function: in words, with a table, or with a formula. But we can also use a recursive formula to describe the operation of the function. eg.

$$f(n) = \begin{cases} 1, \text{if } n = 1 \\ n.f(n-1), \text{if } n > 0 \end{cases}$$

This definition does not tell us right away how to find the value of $f(n)$. $f(n)$ is described in terms of $f(n-1)$ (except for one simple *base case*, $n = 1$). From the recursive formula we can see that $f(1)$ = **1**. It is easy to work out that $f(2)$ = **2** *(as on the second line we have 2 . $f(2\text{-}1)$ = 2 . $f(1)$ = 2 . 1 = **2**)*. Next: $f(3)$ gives 3 . $f(3\text{-}1)$ = 3 . 2 = **6**. Next $f(4)$ gives 4 . $f(4\text{-}1)$ = 4 . 6 = **24**. And so on, eventually you will notice the formula is calculating factorials, ie. $f(n)$ = n!

A recursive function to find an item in a list might defined by the following:

$$listSearch(n, m) = \begin{cases} \text{f}, \text{if m} = [] \\ \text{t}, \text{ if n=m}[0] \\ \text{listSearch(n,m}[1:]) \end{cases}$$

There are 2 base cases in this formula, false if we get to the end of the list or true if we find the item, *(otherwise we keep searching)*.
Note: In reality the listSearch function does the same job as the Python "in" operator (ie listSearh(n,m) <=> n in m) and the in operator is more efficient, so you would normally use that unless for very good reasons.

Example 1: Write a recursive function that raises a value n to the tenth power for a non-negative integer.

```
def pow10(n): if n == 0: # base case return 1 else: return pow10(n-1) * 10
```

Example 2: Write a recursive function that takes a string and returns a reversed string?

```
def reverse(s): if len(s) > 1: s = reverse(s[1:]) + s[0] return s
```

Recursion in the wild
For the vast majority of computer problems iteration is enough. However there are some problems where recusion is the more elegant, certain kinds of problem solving such as "Towers of Hanoi" puzzle or processing a recursive data structure such as a tree. For instance a computer file system is organised as a tree to process files (searching, renaming, moving, etc) often involves a recursive transition of the tree filing system.

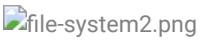

Figure 2: *The "tree" of a computer filing system*

Figure 3: *The Towers of Hanoi puzzle*

Now try to write the Python code to answer the following questions in the cells below. *You may use Python built-in or library functions that might help but **not** third-party libraries such as Numpy.*

1. The following code could be used to implement the factorial function:

```
def fact(n): return fact1(n, 1)
```

```
def fact1(n, r): if n <= 1: return r return fact1(n-1, n*r)
```

which part of it (if any) is recursive and why? *Identify the components that make its recursive.*

fact1 is recursive due to it calling itself with value of n being reduced and updating r value.

```
1
```

2. The recursive definition of *s(n)* is given as

$$s(n) = \begin{cases} 1, \text{if} n = 1 \\ s(n-1) + n, \text{ if} n > 1 \end{cases}$$

Implement this definition as a Python function

```
def s(n):
  if n == 1:
    return 1
  return s(n-1)+n
s(2)
```

```
3
```

3. Write and test a recursive function

```
def printDigits(n):
```

that displays a triangle with n rows, made of digits. For example, *printDigits(5)* should display

```
55555 4444 333 22 1
```

The function's code, under the def line, only needs about three lines of code. *Hint: print(n*str(n)) prints 'n' n times.*

```
def printDigits(n):
  if n < 1:
    return
  print(n*str(n))
  return printDigits(n-1)
printDigits(5)
```

```
55555
4444
333
22
1
```

4. The same as Question 2, but invert the triangle, so that *digitsPrint(5)* prints

```
1 22 333 4444 55555
```

Hint: the solution is in how you manipulate the recursion.

```
def printDigits(n):
  if n > 5:
    return
  print(n*str(n))
  return printDigits(n+1)
printDigits(1)
```

```
1
22
333
4444
55555
```

5. Write a non-recursive function, *called sumDigitsi*, that takes a positive integer and returns the sum of the digits in the integer. i.e. sumDigitsi(16) = 7, sumDigitsi(111) = 3. *Hint: A while-loop is recommended to process the digits*.

```
def sumDigits(n):
  rem,sum = 0,0
  while(n >0):
    rem = int(n%10)
    n =int(n/10)
    sum+=rem
  print(sum)
sumDigits(888)
```

```
24
```

6. Rewrite the solution from question 5) this time as a recursive function *(call it sumDigitsr)* that returns the sum of the digits in a positive integer. i.e. sumDigitsr(42) = 6, sumDigitsr(1234) = 10.

```
def alan(n):
  if n <= 0:
    return n
  rem = int(n%10)
  return rem + alan(n/10)
alan(100)
```

```
1.0
```

7. The listSearch function (as defined in the preamble) is able to find items on a flat list. However, neither listSearch or the in operator are able to find items in a tree data structure such as [1,2,[3,[4],5],6,[7,8]]. So write a new recursive Python function that is able to find items on a tree, such that:

```
treeSearch(4, [1,2,[3,[4],5],6,[7,8]]) => True
treeSearch(9, [1,2,[3,[4],5],6,[7,8]]) => False
```

Test your solution to ensure it accurately returns True or False.

Hints: the function is likely to be similar to the listSearch, you may need to use the function isinstance(< object >, list) and some of the boolean algebra, that you learned last week, may be useful.

```
def treeSearch(item,tree):
  if item == tree:
    return True
  elif isinstance(tree,list):
    for element in tree:
      if treeSearch(item,element):
        return True
  return False
print(treeSearch(4, [1,2,[3,[4],5],6,[7,8]]))
print(treeSearch(9, [1,2,[3,[4],5],6,[7,8]]))
```

```
True
False
```

Please note even if you are using Jupyter on your computer (ie. not on a network) the Jupyter notebook runs as a server, so you are editing and running a file that is not easily accessed on your filing system. So to obtain your workbook (to use on a different computer or upload as an assignment, etc.) it must be downloaded to your file system. To do this...