CMPUT 175 - Lab 8: Recursion

Goal: Practice writing recursive functions.

* You can use your editor's debugger to trace though the recursive function calls on a call stack. (https://eclass.srv.ualberta.ca/mod/url/view.php?id=3929937)

Exercise 1:

Write a **recursive** function *mylen(some_list)* that determines the length of a list, some_list, passed as an argument to this function. Call the function by writing something similar to the following:

```
def main():
    alist=[43,76,97,86]
    print(mylen(alist))
main()
```

Sample Output:

```
4
```

NOTES:

1) The function mylen should work with a list containing any kinds of objects.
2) Your function cannot call the built-in Python function len()!

Exercise 2:

Write a **recursive** function called *intDivision(dividend, divisor)* that uses recursive subtraction to find the integer result of dividend/divisor. Call the function by writing something similar to the following:

```
def main():
    n = int(input('Enter an integer dividend: '))
    m = int(input('Enter an integer divisor: '))
    print('Integer division', n, '//', m, '=' intDivision(n,m))
main()
```

Sample Output:

```
Enter an integer dividend: 65
Enter an integer divisor: 12
Integer division 65 // 12 = 5
```

NOTES:

1) Your function should verify the validity of the dividend and divisor inputs. Both should be positive integers; the dividend can also be 0, but the divisor cannot. Raise an exception if either of the inputs are invalid.
2) **HINT :** Subtracting the divisor from the dividend will eventually yield a value that is less than the divisor. Knowing this, what should your base case be?

OPTIONAL CHALLENGE: modify *intDivision* so that it can accept and handle negative dividends and divisors as well.

Exercise 3:

Write a **recursive** function that computes and returns the sum of digits of an integer. Call the function by writing something similar to the following:

```
def main():
    number = int(input('Enter a number:'))
    print(sumdigits(number))
main()
```

Sample Output:

```
Enter a number: 78411
21
```

NOTE:

1) Your function should raise an exception if the user does not provide a **positive** integer number.

Exercise 4:

Write a **recursive** function that displays the digits of an integer value in reverse order on the console. For example a call to *reverseDisplay(12345)* should display 54321. Call the function by writing something similar to the following:

```
def main():
    number = int(input('Enter a number:'))
    reverseDisplay(number)
main()
```

Sample Output:

```
Enter a number: 73625
52637
```

NOTE:

1) Your function should raise an exception if the user does not provide a **positive** integer number.

Lab continued with Exercise 5 on next page…

Exercise 5:
Consider the following **non-recursive** solution of **binary search** that finds and returns the position of the key in alist, or returns ‘Item is not in the list’ if key is not in the list:

```
def binary_search1(key,alist,low,high):
    # finds and returns the position of key in alist
    # or returns ‘Item is not in the list’
    # - key is the target integer that we are looking for
    # - alist is a list of valid integers that is searched
    # - low is the lowest index of alist
    # - high is the highest index of alist
    found = False
    while (not found and low<=high):
        guess = (high+low)//2
        if (key == alist[guess]):
            found = True
        else:
            if (key < alist[guess]):
                high = guess - 1
            else:
                low = guess + 1
    if (not found):
        guess = 'Item is not in the list'

    return guess
```

Sample calls to binary_search1:

```
def main():
    some_list = [-8,-2,1,3,5,7,9]
    print(binary_search1(9,some_list,0,len(some_list)-1))
    print(binary_search1(-8,some_list,0,len(some_list)-1))
    print(binary_search1(4,some_list,0,len(some_list)-1))
main()
```

Sample Output:

```
6
0
Item is not in the list
```

Your task is to write a **recursive** binary search function called *binary_search2* that will do the same task that is done by *binary_search1* i.e. if binary_search1 in the above sample calls is replaced with *binary_search2*, it would produce the same output. The parameters for binary_search2 must be same as binary_search1.

NOTES:

1) You can assume that the key will always be an integer number.
2) You can assume that alist only contains integers.