Q10)

No both the functions do not give the same answer . in the firt function (A) it checks whether the number is even and if it is then it doubles the number and if it is odd then it adds one to it but it is not inserted in the list so basically it does nothing.There is a issue that if an even number is reached then it inserts twice of it but moves the old number an index ahead so the function checks the next number which was the old number which came an index ahead so it is an infinite list.

Example ;

Input [1,2,3,4]

Output :

[1,4,4,4,4,4,4,4,4,4,4,4,4 -- -- --- -- - --- - - -- - - --infinite times]

In function (B) it checks if the number is odd then it adds one to it at that position and if the number is even then it doubles it and inserts it at its old index so the next time the loop works it takes the old number again and does the same operation on it but it is not infinite list as the length of the list it has take initially is fixes ==len of initial list

Example :

Input :

[1,2,3,4]

Output:

[2,4,4,4,2,3,4]

2.i. error

ii. ((), {1: [5]}, [], 5)

iii . error

iv . ((), {}, [5], 5)

v. ()

vi. error

3. Yes option(b) and option(d) give the same result as in option b it first goes to -1 position which is 40 and then at the position just before 40 as 3 rd position is 40 so (

positon of the cursor : [10,20,30, (I)40]) so at this position it adds 5*[0] and gives the result [10,20,30 ,0,0,0,0,0,40]

And with option d starts from position -1 which is 40 and then tries to go further right but the final position given is 20 which is at the left so it stays there only and replaces 40 by 0,0,0,0,0 final result==[10,20,30,0,0,0,0,0,40]

4.

Firstly (if not lst) check is not required as if the list is empty then the last statement will return an empty list by it self so this was not needed

Secondly (if k not in lst) is a wrong check as it checks if there is an element in the lis whose value is == k which is not we are trying to check.

Thirdly (k=k% (len (lst) - 1)) is a wrong check as if we do this then we will be rotating the items of the list only length of list -1 times only so we have to write this (k=k% (len (lst)) instead of the previous line of code.

5. the code is checking if the data is a list or a tuple and if the data is a list then it creates a nested list and if it is tuple then it creates a highly nested tuple

a. [[[[[[]]]]]]

b. ((((((),),),),),),)

no the answer of a and b will not remain the same if we change the second parameter as the second parameter decides the number of times the data has to be nested.

No the condition k<=0 is not redundant as if this condition will not be there then ther will be an recursion depth exceeded error as then the function will call itself infinite number of times.

6. output : {1: True, 2: False, 3: 'b'}{2: False, 3: 'b', 1: True}

7. a) will pass

```
def test():
        assert  sortList([1,2,3,4,5])==[1,2,3,4,5]
```

b)will not pas:

```
def test():
        assert  sortList([4,2,3,4,5])==[2,3,4,4,5]
```

c) at line number 8 instead of this condition (while j >= 1 and key < a[j]:)

the condition should be (while j >= 0 and key < a[j]:)

8.

A . 1) assert expr_conv("a+b*c-d/e") == "abc*+de/-"

2) assert expr_conv("a*(b+c)/(d-e)") == "abc+*de-/"

3) assert expr_conv("a+b*(c-d)/e^f") == "abcd-*ef^/ +"

b) the programmer can check that the input is correct by checking that after every letter an operation like " - + / * ^ " are used .