1.
update_listA is a function that takes a list as a parameter. If an element in the list is even, then it multiplies the element by two, inserts it before the element else adds one to the element, and prints the list on every iteration. Since the size of the list is increasing at every iteration when at least one element is even thus forming an infinite loop.

update_listB is a function that takes a list as a parameter and iterates over its original length. If an element in the list is even, it multiplies the element by two and inserts it before that element; otherwise, it adds one to the element and prints the list on every iteration.
No, the result of both the functions are not the same as it is explained above the result of update_listA will be an infinite loop, and update_listB will be [2,4,4,3,3,4].

2.
a. It gives an error because tuple has no attribute append.
b. ((), {1: [5]}, [], 5) - This is the output because we first give the dictionary a value of an empty list for key 1 and then append on that list.
c. It gives an error because tuple has no attribute append.
d. ((), {}, [5], 5) - this is because we are appending 5 to a list
e. () - this is because we are changing tup1 to point to tup
f. It gives an error because tuples are immutable.

3.
Yes, b and d have the same output because slicing indexes [-1:3] and [-1:-3] are empty lists.

4.
Useless pieces of code are as follows:
if k not in lst:
return lst
This code returns an unchanged list if k is not in lst, which is not needed.
Error:
k = k % (len(lst)-1)
It should be changed to:
k= k % len(lst)
Otherwise, the list remains unchanged as 12%4=0

5.
The code is a recursive function which recursively inserts a given list inside empty k times if data is a list, and it tries to do the same for tuples. Still, as tuples are immutable, we get the original tuple.
a. [[[[[[]]]]]]
b. ()
- The answer to b remains the same even if we change the second parameter, but the answer for a changes when we change the second parameter.
- The first condition is not redundant because we will have an infinite number of recursions

without that condition.

6.
{1: True, 2: False, 3: 'b'} {2: False, 3: 'b', 1: True}
- The first two lines are assignment lines
- Third line checks the condition of dict_a[2] not equal to string ‘dict_b[2]’ which is true, thus dict_b[1] gets value True
- The fourth line checks the condition of dict_a[2] equal to dict_b[2], which means ‘b’ is equal to ‘c’, which is False, dict_b[2] gets the value False
- The fifth line updates the key-value pairs to dict_a to same as dict_b
- The sixth line prints dict_a and dict_b.

7.
a. assert sortList([1,36,2,6])==[1,2,6,36]
b. assert sortList([10,52,26,2])==[2,10,26,52]
c. In line 8:
while j>=0 and key<a[j]:
We do this because otherwise if the last number is the least, it will not be moved to the starting.

8.
a. 1. assert expr_conv(‘a*b-c/d+k’)==’ab*cd/-k+’
2. assert expr_conv(‘(a+c)/(b+d)*e-f’)==’ac+bd+/e*f-’
3. assert expr_conv(‘(a+b*c)-d/f^e’) ==’abc*+dfe^/-’
b. Input should be validated before being processed.