For questions 1 and 2 the answers are pretty definite. For questions 3 and 4 there are a number of ways one could do a working program. We will accept alternatives, taking off points on a working program only if it is much less efficient than the suggested answer.

1. The functions fee and foo, 1a and 1c respectively, return the given string in reverse order. The function fie, 1b, causes a run time TypeError because of the attempt to concatenate an int to a str in the next to the last line. The index in I should be used to select a character from the argument string, x, as

```python
def fie(x):
    y=""
    for i in range(len(x)):
        y=x[i]+y  #corrected from y=i+y
    return y
```

The function fum, 1d, has a logical error. It uses negative position values to count forward from the last position to the first, but starts incorrectly with 0 instead of -1. One could either have the range(1,len(x)+1) in the for command or simply subtract 1 from the selector index in the concatenate command, as

```python
def fum(x):
    y=""
    for i in range(len(x)):
        y=y+x[-i-1]   #corrected from y=y+x[-i]
    return y
```

2. These are pretty straight forward:
   a. decimal 2 is signed binary 00000010 and hexadecimal 0x02
   b. decimal -2 is signed binary 11111110 and hexadecimal 0xfe
   c. decimal 42=32+8=2*16+8 is signed binary 00101000 and hexadecimal 0x28
   d. decimal 127 is signed binary 01111111 and hexadecimal 0x7f
      This is the largest positive signed binary number for 8 bits.
      If you didn't know that you could decompose it as 64+32+16+8+4+2+1.
   e. decimal -127 is signed binary 10000001 and hexadecimal 0x41
      Just negate, 2s complement, the result for part d
   f. decimal 65=64+1=4*16+1 is signed binary 01000001 and hexadecimal 0x41

3. Name   Op  ADD1  ADD2  ADD3      Comments
   start  lc  r3         0az      ASCII z to r3 for compare
   lc  r4         0aa      ASCII a to r4 for compare
   lc  r5         notlc    found not lower case address
   loop   in  p2    r2            input ASCII character
   bgt r5    r2    r3      is its code > that of z?
   bgt r5    r4    r2      is its code < that of a?
   jmp             loop    if not, go back to get next char
   notlc  out p3    r2            if it is, then out put the character
   hlt             start   and halt
4a. def distance(s1,s2):
    count=0   #the number of differences will be added up in count
    i=0
    while i<len(s1):   #for every char position in the strings
        if s1[i]!=s2[i]:  #If they differ
            count=count+1   #count it
        i=i+1
    return count

4b. def closest(s,slist):
    if len(slist)==0:   #if empty list return -1
        return -1
    else:  #else guess that the closest string is in position 0
        closest_so_far=0
        distance_so_far=distance(s,slist[0])
        for i in range(1,len(slist)):  #for every position in slist after 0
            distance_to_i=distance(s,slist[i])
            if distance_to_i<distance_so_far:  #if slist[i] is closer than best guess
                closest_so_far=i   #improve the guess
                distance_so_far=distance_to_i #and save the distance for comparison
        return closest_so_far   #return the updated guess

    #OR, less efficient since it calls distance more often, but simpler and prettier is
    def closest2(s,slist):
        if len(slist)==0:   #if empty list return -1
            return -1
        else:  #else guess that the closest string is in position 0
            besti=0
            for i in range(1,len(slist)):  #for every position in slist after 0
                if distance(s,slist[i])<distance(slist[besti],slist[i]):  #if slist[i] is closer than best guess so far
                    besti=i   #improve the guess
            return besti   #return the best guess after trying them all

4c. def remove(s,slist):
    newlist=[]  #the strings that pass the test will be appended here
    for s2 in slist:
        if distance(s,s2)<=5:   #if no more than 5 chars different
            newlist=newlist+[s2]  #save it
    return newlist