1. Write a function that meets the following specifications.

```
def is_power(x, p):
    """
    x: an int
    p: an int > 0
    Returns True if there exists a positive int, i, such that
    p**i == x, and False otherwise.
    """
    # your code here
```

For example:

```
print(is_power(4,2))   # prints True
print(is_power(5,2))   # prints False
print(is_power(-4,2))  # prints False
print(is_power(9,3))  # prints True
print(is_power(15,3))  # prints False
print(is_power(3,1))  # prints False
```

2. Write a function that meets the following specifications.

```
def occurs_most_often(L):
    """
    L: a list of ints
    Returns the element in L that occurs the most often in L.
    In case of a tie, returns the maximum of the tied values.
    Raises a ValueError if L is empty.
    """
    # your code here
```

For example:

```
print(occurs_most_often([1,2,1]))          # prints 1
print(occurs_most_often([6,6,2,2]))        # prints 6
print(occurs_most_often([6,0,6,2,0,2,0]))  # prints 0
print(occurs_most_often([]))  # raises a ValueError
```

3. Write a function that meets the following specifications.

```
def max_contig_subseq_sum(L):
    """
    L is a list of integers, at least one positive
    Returns the maximum sum of a contiguous subsequence in L. A contiguous
    subsequence is made up of consecutive elements. It can be of any length,
    and start at any point.
    """
    # your code here
```

For example:

```
print(max_contig_subseq_sum([3, 4, -1, 5, -4]))   # prints 11
print(max_contig_subseq_sum([3, 4, -8, 15, -1, 2]))  # prints 16
```

4. Write a function that meets the following specifications.

```
def decode_ints(s_from, s_to, L):
    """ s_from, s_to are lowercase strings of equal length, each
                    only containing character digits 0-9 in some order
        L is a list of positive ints

        Returns a tuple of (map, L_decoded) where:
        * map is a dict of 10 keys mapping an int digit in s_from at index i
          to a corresponding int digit in s_to at index i.
        * L_decoded is list of decoded ints using the map, where each digit
          in the int is mapped to its corresponding digit from the map
          (ignore all leading zeroes in decoded ints. ) """
    # your code here
```

For example:

```
s_from = "0123456789"
s_to =   "9876543210"
L = [1, 5, 123]
print(decode_ints(s_from, s_to, L))  # prints a tuple:
# ({0:9,1:8,2:7,3:6,4:5,5:4,6:3,7:2,8:1,9:0}, [8, 4, 876])

s_from = "1234567890"
s_to =   "0234567891"
L = [0, 6719]
print(decode_ints(s_from, s_to, L))  # prints a tuple:
# ({1:0,2:2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,0:1}, [1, 6709])
```

5. Write a function that meets the following specifications.

```
def elems_matching(L1, L2, f):
    """ L1 is a non-empty list containing unique ints
        L2 is a non-empty list containing ints, of the same length as L1
        f is a function that takes in an int and returns an int

    Returns a dictionary that maps an element at index i from L1 to an
    element at index i in L2. The only key-value pairs in the dict are
    those at index i where f applied to L1 at index i equals L2 at index i.
    """
    # your code here
```

For example:

```
L1 = [1,2,3]
L2 = [2,3,2]
def f(a):
    return a+1
print(elems_matching(L1, L2, f))  # prints {1: 2, 2: 3}

L1 = [0,3,7,9]
L2 = [0,0,0,9]
def f(a):
    return 0
print(elems_matching(L1, L2, f))  # prints {0: 0, 3: 0, 7: 0}
```

6. Write a function that meets the following specifications.

```
def is_formed_by_s(s1, s2):
    """ s1 and s2 are strings containing lowercase alphabetic characters
        Returns True if s2 can be formed using the letters in s1
        in the order in which they appear in s1. """
    # your code here
```

For example:

```
print(is_formed_by_s('abcd', 'bcd')) # prints True
print(is_formed_by_s('abcd', 'ad')) # prints True
print(is_formed_by_s('abcd', 'ba')) # prints False
```

7. Write a function that meets the following specifications.

```
def mutate_if_formed_by_L(L1, L2):
    """ L1 and L2 are lists containing ints
        Determines if the elements in L2 can be formed using the
        elements in L1 in the order in which they appear in L1.
        If yes, mutates L1 to contain the same elements in L2.
        If no, L1 stays unchanged.
        Returns None."""
    # your code here
```

For example:

```
La = [1,2,3,4]
Lb = [2,3,4]
mutate_if_formed_by_L(La, Lb)
print(La)   # prints [2, 3, 4]

La = [1,2,3,4]
Lb = [1,4]
mutate_if_formed_by_L(La, Lb)
print(La)   # prints [1, 4]

La = [1,2,3,4]
Lb = [2,1]
mutate_if_formed_by_L(La, Lb)
print(La)   # prints [1, 2, 3, 4]
```