3.

fun func (x:'a) (y:'b) = fn (c:'b) => x;

5.

f = lambda x: [[y for j, y in enumerate(set(x)) if (i >> j) & 1] for i in range(2**len(set(x)))]

Lambda is basically a fast way to write a function. x is its parameter.

There are 2 list comprehensions in here, meaning that the output is a list of lists.

The outer one creates a list element, for every number in range of 0 to 2 to the power of the length of x (the input parameter). Before all that, x is turned into a set, which means that x is a collection of some sort (list, tuple, dictionary) and that all duplicates in x are removed.

The inner one takes x (the set), enumerates every element and iterates through it. Enumerate basically creates tuples of each element (j,y) – j is the element created by the enumeration. The loop keeps going until there are tuples still left, which there should be. The list comprehension only takes y into its list, but this list is also filtered.

There is an if statement (i >> j) & 1. This means that i's bits are shifted to the right j amount of times, and then AND is applied to that output. & 1 stands for 01 in binary and will give output of 1 if the other number has this "pattern" of bits. In this case, these are 1,3,5,7,9,11,13 – all odd numbers. Each has 01 of its binary digits.

Now for a more practical approach.

If i=0, then the output will be [ ] – an empty list. Because no matter what, 0>>x will always be 0 and 0 & 1 will be 0 (false).

If i=1, then the output will be [first element of set x]. Because j starts off with 0 and x>>0 will be x.

It's also important to note >> doesn't update the variable, meaning that i doesn't change.

Here is what I am going to assume. In a list that has 3 elements, the first loop will loop 8 times – meaning a list of 8 lists. Range will go from 0 to 7 – values of i. The inner loop is limited to j, which is limited to the amount of elements in set x. J will be from 0 to len(x)-1.

In this case, j is equal to 0,1,2 - meaning that with 3 elements it will do 3 loops.

Then I went a bit further and used IDLE.

```
>>> l = list([1,2,3])
>>> f(l)
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
```

I deduced that indexes 1,2,4 give only one value. I figured that it is because they are the powers of 2 and are used to convert binary to decimal and back. Then it hit me, (1,2,4) only have one 1 representing the decimal. One 1 = one element. Meanwhile something like 7, which is represented as 111 in binary has 3 elements.

>> checks every binary digit and **& checks if** that digit is **equal to 1.**

Something like 5 (101) gives 2 elements – 1 and 3. While 6 (110) gives 2 and 3. **Every digit represents an element** in here. I guess this is because i is taken from range of 0 and 2 to the power of len(x) **– keyword being 2 to the power**.