Every tile has a different tile number (index). Assuming that you number the tiles in the natural way, the tiles in the first tiling will run from 0 to 120, and the tiles in the second tiling will run from 121 to 241 (why?).

Math. Each tiling contains 121 tiles, and we sequence the tilings continuously. So the next group of tiles (a tiling) starts where the previous one ends and continues for 121. Thus. 121->241

A given `input` point will be in exactly one tile in each tiling. For example, the point from the first example in the training set above, in1=0.1 and in2=0.1, or 0.1,0.1, will be in the first tile of the first seven tilings, that is, in tiles 0, 121, 242, 363, 484, 605, 726 (why?). The point 0.1 is essentially 0.1/0.6 = 1/6th the length of a tile up and to the right of the tile origin. *Each new tiling effectively shifts the point 1/8th a tile. So for the 7th tiling, it has been shifted (7-1)/8th = 0.75 of the way to the origin. 7-1 because no shifting takes place in the first tiling. So still has not shifted enough to consume the point 1/6th = 0.17 away from the origin.. Therefore it's in the first tile.* In the eighth tiling this point will be in the 13th tile (why?), *By the 8th tiling, the point has been shifted by 7X1/8th = 0.875 which means that the point at 0.17 would no longer fall in this first tile. It be shifted up and to the right tile* which is tile 859 (why?) *The first tile of this tiling would be 847. But this point doesn't fall into that. It falls up a row and to the right a column. So 11+1 = 12 tiles beyond. So 847+12 = 859.*
`tilecode(0.1,0.1,tileIndices)`, then afterwards `tileIndices` will contain exactly these eight tile indices. The largest possible tile index is 967 (why?). *121 tiles / tiling X 8 tilings = 968 possible tiles. But 0 index based. So max index of 967*.

Part 2

Run SuperLearn.py. The last line of the file calls the test1 function which calls your learning algorithm on the four example points and prints out results. As a result of learning, your approximate function value should move 10% of the way from its original value towards the target value. The before value of the fourth point should be nonzero (why?) *The fourth point is non-zero because it is in similar to the second point - which has already been learned about and is thus non-zero. This is demonstrates the power of generalization.*

You should see the MSE coming down smoothly from about 0.25 to almost 0.1 and staying there. *Why does it not decrease closer to 0? It doesn't continue to decrease further because the target function itself is not deterministic. It has a variance of 0.1. So a function that perfectly predicted the future value would still be on average off because of this variance.*

After only 20 examples, your learned function will not yet look like the target function. Explain in a paragraph why it looks the way it does. If your learned function involves many peaks and valleys, then be sure to explain both their number, their height, and their width.

When learning starts, the 3D function graph would look flat since the estimates for weights are all 0. From here, I like to consider the effect of new, incoming training data. Processing training data (a new input/output example) is visualized by 1. pinching the graph at the input value and then 2. pulling it up (or down) based on the output value. Depending on what the step size is, the graph will stretch part or all of the way to where you stretch it. ie.

a step size of 1 will cause the graph to "stick" where you let go. While 0.5 will result in it recessing 0.5 of the way back. Also, it isn't just the point at which you pinch the graph that is stretched. It's the area around it. The "area around it" is a function of what the symmetry of the tilings/features. Wide tilings mean that the tiles are wide, and thus, a wide range of surrounding areas of the graph are pulled up with the pinch. With this model in mind, it is easy to see that each bit of training data will cause a "peak" or value. The height depends on what the target function says the value should be (and how often it's been pulled in that area of the graph). Obviously if input values are close together, they generalize and the peaks combine so that it may look like one peak (or valley). Also, if the target function returns 0, the training data wouldn't change the graph much, so neither peak or valley would be obvious.

Suppose that instead of the tiling input space into an 11X11 grid of squares, you had divided into an 11X21 grid of rectangles with the in1 dimension being divided twice as finely as the in2 dimension. Explain how you would expect the function learned after 20 examples to change if this alternative tiling were used.

An 11X11 grid is uniformly offset tilings. For these types of offsets, generalizations happen along the diagonal of input space. With asymmetric offsets (which would be the case with an 11X21 grid), the generalizations are more spherical. Furthermore, if the dimension is defined more finely, the generalizations won't be so broad. Therefore, the peaks and valleys on the graph won't be as wide. Furthermore, where two different inputs in the past may have combined to form a peak, they may now form 2 distinct peaks since they no longer overlap.