Answer: In the notes we considered the LMS algorithm for fitting the model $h_\theta(x) = \theta^T x$ with update rule $\theta := \theta + \alpha \sum_{i=1}^n (y_i - h_\theta(x_i))x_i$. Here $x \in \mathbb{R}^d$, and in order to introduce a feature map $\phi(x) \in \mathbb{R}^p$ e.g. $(1, x_1, x_2, x_1^2, x_1x_2, ...)$ etc., we had then $\theta := \theta + \alpha \sum_{i=1}^n (y_i - \theta^T\phi(x_i))\phi(x_i)$.

Applying the Kernel trick in this scenario to deal with high/infinite dimensional feature involved taking $\theta^{(0)} = \sum_{i=1}^n \beta_i^{(0)} \phi(x_i)$ and observing that the update becomes $\beta := \beta + \alpha(y - K\beta)$ where K is the matrix $K_{ij} =< \phi(x_i), \phi(x_j) >$, and we've "replaced" θ with a new parametrisation of the n-vector β.

Then resultant prediction $\theta^T\phi(x) = (\sum_j \beta_j \phi(x_j))^T \phi(x) = \sum_j \beta_j K(x_j, x)$. The important is that we've replaced keeping track of high dimensional $\phi(x)$ by instead computing easier $K(x, z) = \phi(x)^T\phi(z)$ which generally should be easier to compute.

In the perceptron learning algorithm we have a slightly different setup: $h_\theta(x) = \text{sgn}(\theta^T x)$, and with update rule $\theta^{(i+1)} := \theta^{(i)} + \alpha(y_{i+1} - h_{\theta^{(i)}}(x_{i+1}))x_{i+1}$. For some reason we make an update based on just one training example, and make just one pass through the training set. $\theta^{(0)} = 0$. The higher-dimensional kernel analogue is then taking at the i+1th step, $\theta^{(i+1)} := \theta^{(i)} + \alpha(y_{i+1} - h_{\theta^{(i)}}(\phi(x_{i+1})))\phi(x_{i+1})$, and so $\beta_{i+1}^{(i+1)} := \beta_{i+1}^{(i)} + \alpha(y_{i+1} - \text{sgn}(\sum_j K_{i+1,j}\beta_j^{(i)}))$.

I.e. at each step we oddly only update one index of β. so $\beta^0 = 0$. Then $\beta_1^1 = \alpha y_1$. Then $\beta_2^2 = \alpha(y_2 + \sum_j \beta_j^1 K_{2,j}) = \alpha(y_2 + \beta_1^1 K_{2,1})$, and so on...