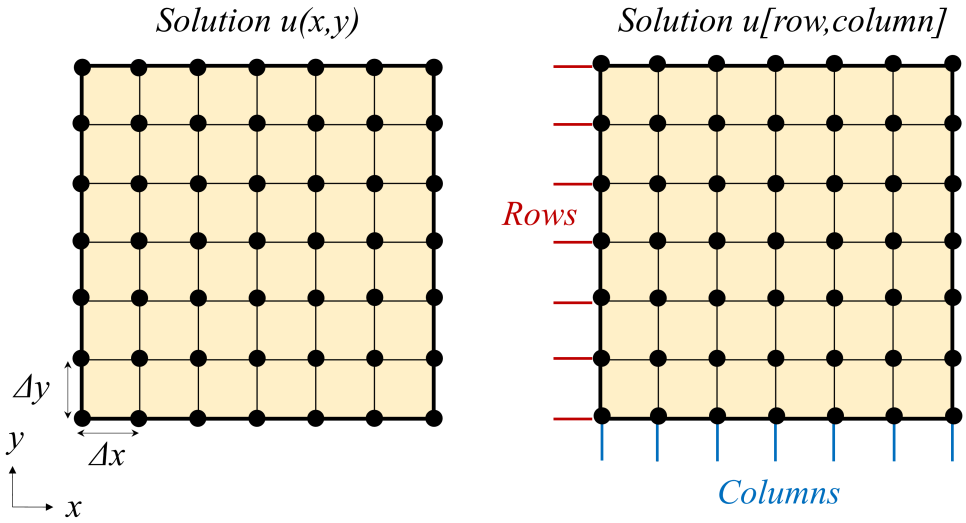
Boundary conditions & how we implement these on arrays in programming

Boundary conditions and solution array

In practice, we have our solution as a function of x and y coordinates u(x,y) and we want to find the solution at all points (x,y). In a computer, this solution will be stored as a 2D array, where each element of the array represents the solution at a point (x,y). Consider the 7 $\times$ 7 grid with discretization steps Δx and Δy in the x and y direction, respectively, shown in the following figure.



On the left the grid is shown with respect to x and y coordinates and on the right as it would be stored in a computer. A 2D array is defined by its rows and columns (essentially it is a table of values) and to access the solution at a specific row and column, we use u[row, column]. For example, to access the solution stored in the 0th row and 4th column, we use u[0,4]. In an array, the columns represent the x direction and the rows represent the y direction and therefore we see that this is opposite order from u(x,y).

Therefore the boundary conditions in the 7 $\times$ 7 grid can be set in an array as:

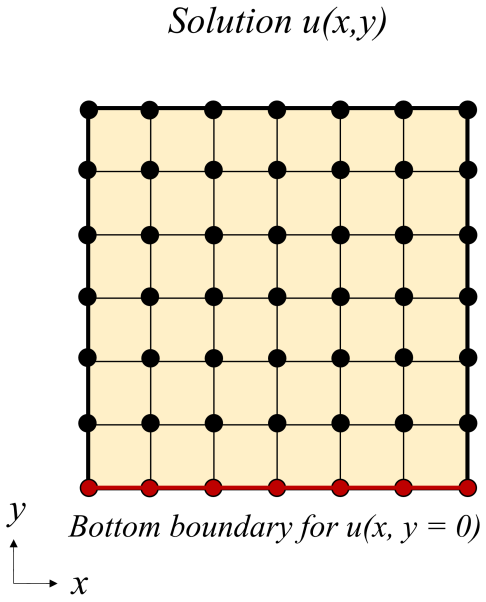
$$
\begin{aligned}
u(x, y=0) &\quad \Rightarrow \quad u[0,:] \\
u(x, y=L) &\quad \Rightarrow \quad u[6,:] \\
u(x=0, y) &\quad \Rightarrow \quad u[:,0] \\
u(x=L, y) &\quad \Rightarrow \quad u[:,6]
\end{aligned}
$$

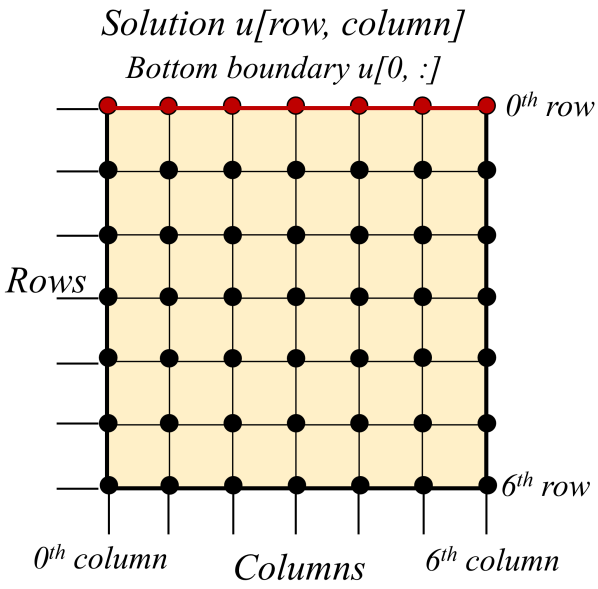
or alternatively in a general grid

$$
\begin{aligned}
u(x, y=0) &\quad \Rightarrow \quad u[0,:] \\
u(x, y=L) &\quad \Rightarrow \quad u[-1,:] \\
u(x=0, y) &\quad \Rightarrow \quad u[:,0] \\
u(x=L, y) &\quad \Rightarrow \quad u[:,-1]
\end{aligned}
$$

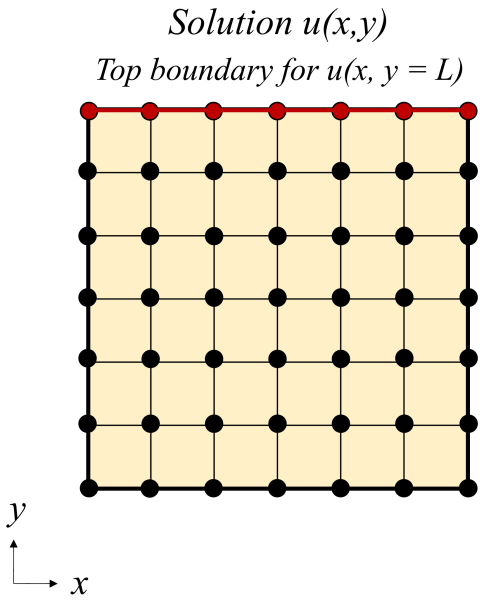
where -1 is the index of the last item of an array/list (counting from the end). The colon ':' means all values in a direction, e.g. u[0,:] takes all column values for fixed row = 0. Similarly, u[:, 0] takes all row values for fixed column = 0. Look the figures below to see which values in the array represent the boundary conditions.

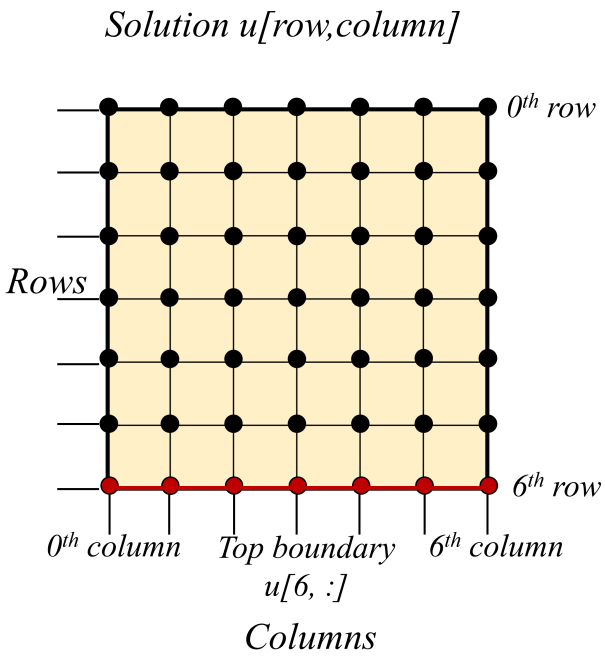
Bottom boundary



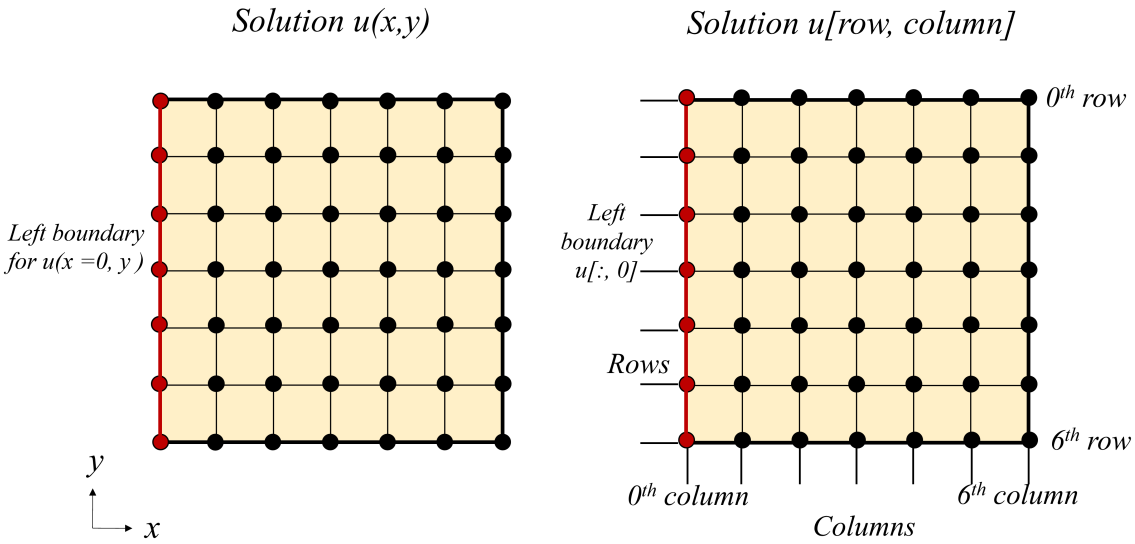


Top boundary

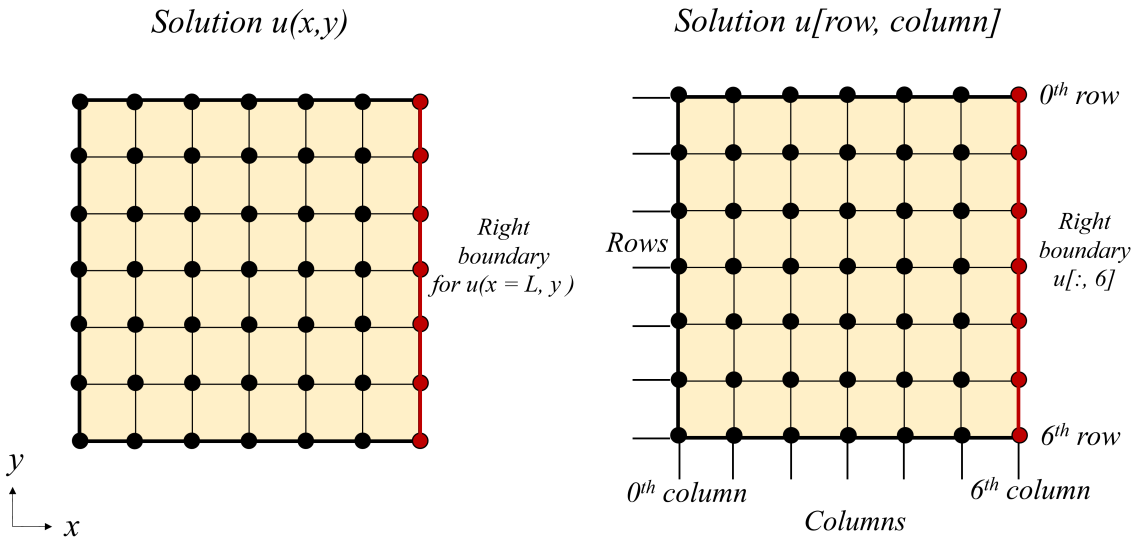




Left boundary



Right boundary



Also, **remember**, in Python we start counting from 0 (not 1)!