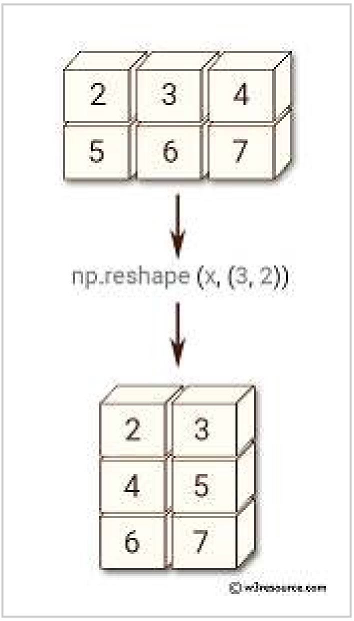
NumPy Basics - Reshaping and Concatenation

WE'LL COVER THE FOLLOWING

5. Reshaping of Arrays

Reshaping is about changing the way items are arranged within the array so that the shape of the array changes but the overall number of dimensions stays the same, e.g., you can use it to convert a *1D* array into *2D* array.

Reshaping is a very useful operation and it can easily be done using the `reshape()` method. Since a picture speaks a thousand words, let's see the effects of reshaping visually:



Reshaping example (Image credits: www.3resource.com)

How can we do this in code? Say we want to create a *3x3* grid with numbers from 1 to 9. We first create a *1D* array and then convert it to the desired shape as shown below.

Run the code in the widget below and **observe the outputs** of the print statements to understand what's going on.

```
import numpy as np

reshaped = np.arange(1, 10).reshape((3, 3))
print(reshaped)
```

Similarly, we can use reshaping to convert between row vectors and column vectors by simply specifying the dimensions we want. A row vector has only 1 row. This means that after reshaping, all the values end up as columns. A column vector has only 1 column and all the values end up in rows.

```
x = np.array([1, 2, 3])
print(x)

# row vector via reshape
x_rv= x.reshape((1, 3))
print(x_rv)

# column vector via reshape
x_cv = x.reshape((3, 1))
print(x_cv)
```

6. Concatenation and Splitting of Arrays

We are often required to combine different arrays or split one array into multiple arrays. Instead of doing this manually, we can use NumPy's array concatenation and splitting operations. This means we can handle these complex tasks easily.

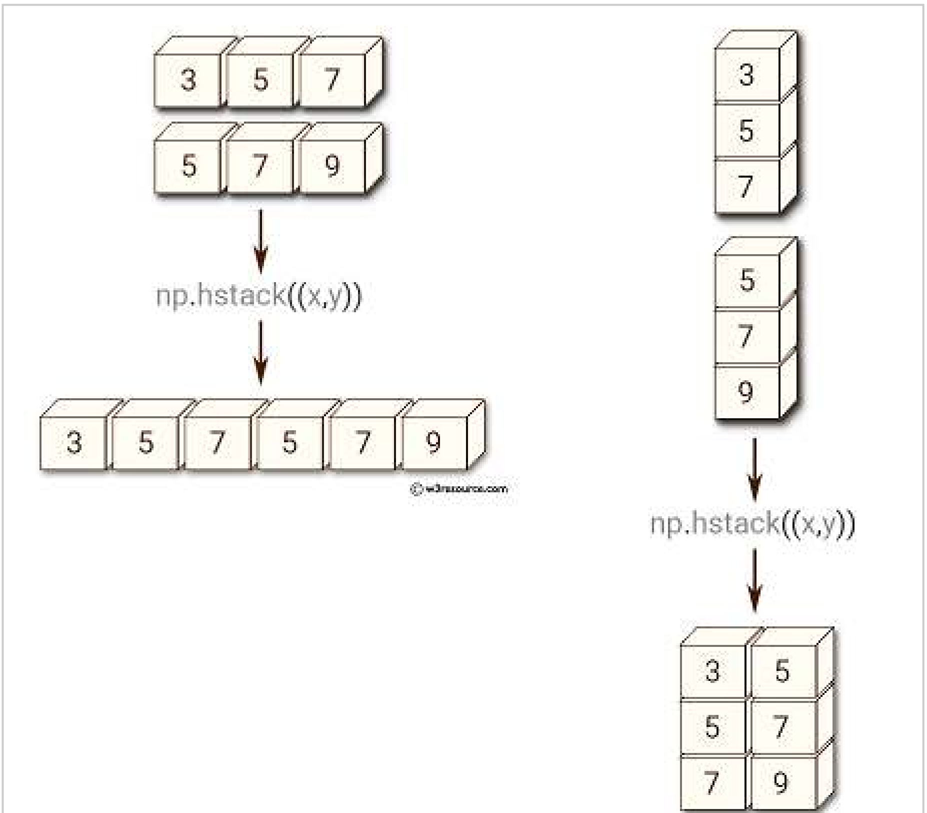
a. Concatenation

The `concatenate()` method allows us to put arrays together. Let's understand this with some examples.

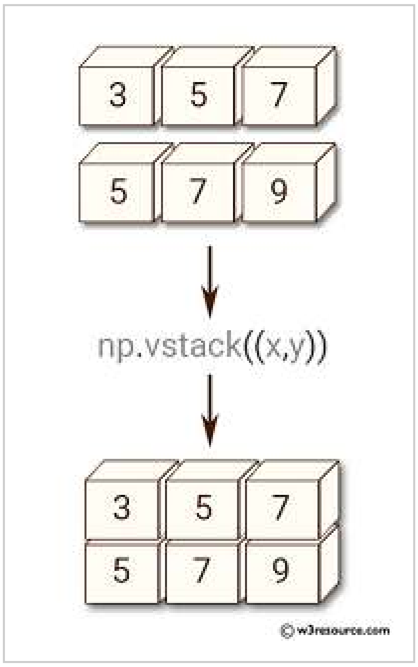
The following code-example first shows how to concatenate three *1D* arrays into a single array and then it shows how to concatenate two *2D* arrays into a single array.

```
# We can concatenate two or more arrays at once.
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
z = [11,11,11]

np.concatenate([x, y, z])
#> array([ 1,  2,  3,  3,  2,  1, 11, 11, 11])

# We can also concatenate 2-dimensional arrays.
grid = np.array([[1,2,3] , [4,5,6]])
np.concatenate([grid, grid])
#> array([[1, 2, 3],
#>        [4, 5, 6],
#>        [1, 2, 3],
#>        [4, 5, 6]])
```

Now, what if you are required to **combine arrays of different dimensions**, e.g., a *2D* array with a *1D* array? In such cases, `np.concatenate` might not be the best option to use. Instead, you can use `np.vstack` (vertical stack) or `np.hstack` (horizontal stack) to finish the task.



Horizontal Stacking (Image Credits: www.w3resource.com)



Vertical Stacking (Image Credits: www.3resource.com)

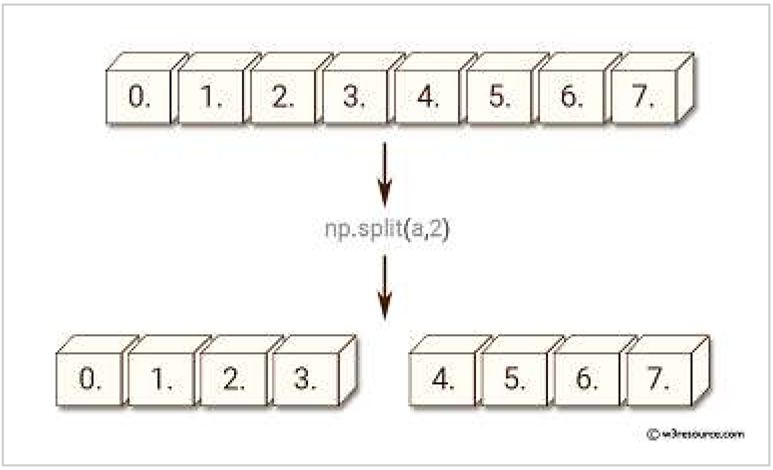
Run the code below to better understand how to do this practically. Before looking at the output, **try to visualize the solution** in your head.

Reminder: You can add a print statement around the output to see results in your console as well.

```
x = np.array([3,4,5])
grid = np.array([[1,2,3],[9,10,11]])

np.vstack([x,grid]) # vertically stack the arrays
#> array([[ 3,  4,  5],
#>        [ 1,  2,  3],
#>        [9, 10, 11]])

z = np.array([[19],[19]])
np.hstack([grid,z])  # horizontally stack the arrays
#> array([[ 1,  2,  3, 19],
#>        [9, 10, 11, 19]])
```

b. Splitting

We can do the opposite of concatenation and split the arrays based on a given position for the split points.



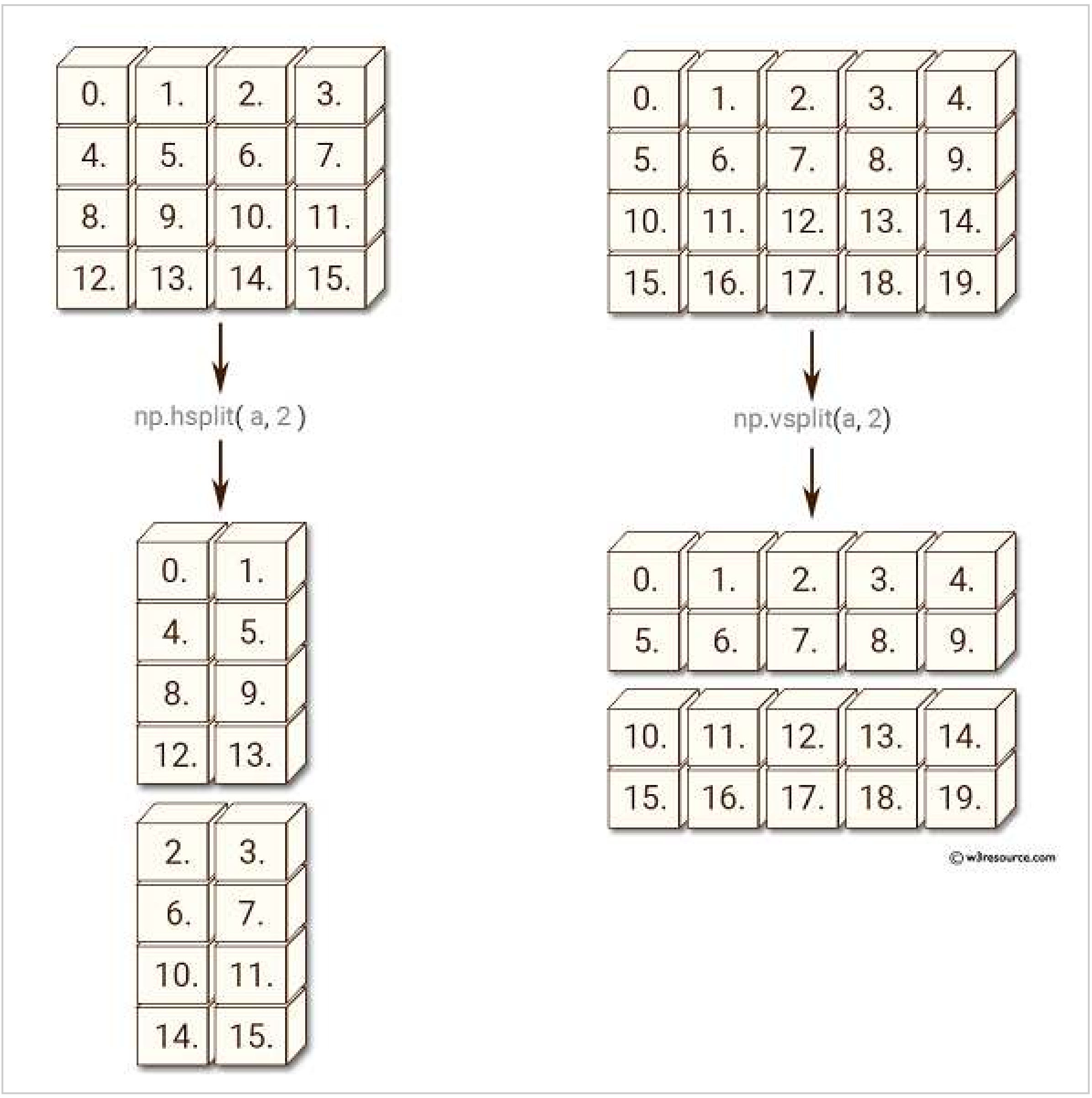
Splitting of an array (Image Credits: www.3resource.com)

We can do it in code like so:

```
x = np.arange(10)
#> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

x1, x2, x3 = np.split(x,[3,6])
print(x1, x2, x3)
#> [0 1 2] [3 4 5] [6 7 8 9]
```

Notice that *N* split points result in *N + 1* subarrays.

Like concatenation, we can also perform **horizontal and vertical splitting**.



Horizontal and Vertical Splitting (Image Credits: www.3resource.com)

Let's see this in action. Run the code below and observe the outputs.

```
import numpy as np

grid = np.arange(16).reshape((4, 4))
print(grid, "\n")

# Split vertically and print upper and lower arrays
upper, lower = np.vsplit(grid, [2])
print(upper)
print(lower, "\n")

# Split horizontally and print left and right arrays
left, right = np.hsplit(grid, [2])
print(left)
print(right)
```

Good job! We are done with the basic operations in NumPy.

In addition to the functions we have learned so far, there are several other very useful functions available in the NumPy library *(sum, divide, abs, power, mod, sin, cos, tan, log, var, min, mean, max, etc.)* which can be used to perform mathematical calculations. There are also built-in functions to compute aggregates and functions to perform comparisons. We are going to learn these concepts in the upcoming lessons, so **keep going!**