KEY_Practice13_Numpy_intro

July 14, 2019

1 Practice with numpy!

Remember: * Numpy provides a bunch of useful tools for performing calculations * You access numpy functions by calling `np.function_name`

First, import numpy. Remember to use the nickname!

[1]:
```
# load numpy
import numpy as np
```

Use numpy to create a list of numbers from 0 through 99. We'll use a function called `arange`.

[2]:
```
# Use np.arrange to generate an array of numbers and assign it to a variable␣
↪called numbers
numbers = np.arange(100)
# print the array to see what it looks like
print(numbers)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
```

Now use numpy's function called `zeros` to create another empty array of the same size:

[3]:
```
# Create an empty array with np.zeros and assign it to a variable
zeros = np.zeros(100)
# print it
print(zeros)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
```

Now try manipulating the arrays using basic math:

[4]:
```
# Add the numbers array to itself
numbers + numbers
```

[4]:
```
array([  0,   2,   4,   6,   8,  10,  12,  14,  16,  18,  20,  22,  24,
        26,  28,  30,  32,  34,  36,  38,  40,  42,  44,  46,  48,  50,
        52,  54,  56,  58,  60,  62,  64,  66,  68,  70,  72,  74,  76,
        78,  80,  82,  84,  86,  88,  90,  92,  94,  96,  98, 100, 102,
       104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128,
       130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154,
       156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180,
       182, 184, 186, 188, 190, 192, 194, 196, 198])
```

[5]:
```
# Multiply the numbers array to itself
numbers * numbers
```

[5]:
```
array([   0,    1,    4,    9,   16,   25,   36,   49,   64,   81,  100,
        121,  144,  169,  196,  225,  256,  289,  324,  361,  400,  441,
        484,  529,  576,  625,  676,  729,  784,  841,  900,  961, 1024,
       1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849,
       1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916,
       3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225,
       4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776,
       5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569,
       7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604,
       9801])
```

Note that when you perform a math operation on an array, it will often perform that operation on each item in that array. It's convenient that we don't have to loop through all the values to apply the math operation to every item in the array.

You can find information about the size of an array by using `.shape`. Note that `.shape` is an **attribute** of array -- a special variable that belongs to every *array object*. Try it out:

HINT: Because `.shape` is not a function you don't need to use parentheses.

[6]:
```
# Use shape to view the size of your array
numbers.shape
```

[6]:
```
(100,)
```

`.shape` gave us just one number, because our array has only 1 dimension. Later we'll see what it looks like for arrays with more than 1 dimension.

Numpy also allows you to create 2D arrays, like with lists. We can use the `method` called `reshape` to change an 1-dimensional `array` into a 2-dimensional `array`. `reshape` takes two arguments: the number of rows and the number of columns. Try turning one of your arrays into a 2D array using `reshape`.

[7]:
```
# Reshape one of your arrays into a 2D array
numbers_2d = numbers.reshape( 10, 10 )
numbers_2d
```

[7]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
```

```
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

Now the `.shape` of your array should be changed, try printing it out below:

[8]:
```
# Print the shape of your new array
numbers_2d.shape
```

[8]:
```
(10, 10)
```

Now we will try a couple of numpy's math functions!

[9]:
```
# try using np.sum to add the items in a list together
print(np.sum(numbers))
```

```
4950
```

[10]:
```
# try squaring the value of each item in your array
print(numbers**2)
```

```
[   0    1    4    9   16   25   36   49   64   81  100  121  144  169
  196  225  256  289  324  361  400  441  484  529  576  625  676  729
  784  841  900  961 1024 1089 1156 1225 1296 1369 1444 1521 1600 1681
 1764 1849 1936 2025 2116 2209 2304 2401 2500 2601 2704 2809 2916 3025
 3136 3249 3364 3481 3600 3721 3844 3969 4096 4225 4356 4489 4624 4761
 4900 5041 5184 5329 5476 5625 5776 5929 6084 6241 6400 6561 6724 6889
 7056 7225 7396 7569 7744 7921 8100 8281 8464 8649 8836 9025 9216 9409
 9604 9801]
```

Try converting the `numbers` array to an array of floats using the method called `astype`:

[11]:
```
# Convert the array into an array of floats
print(numbers.astype(float))
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53.
 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71.
 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89.
 90. 91. 92. 93. 94. 95. 96. 97. 98. 99.]
```

Nice job! You just practiced:

- Using `numpy` to perform `array` operations.
- Performing math with `numpy`.