3_Numpy_1_Solution

September 19, 2019

1 Challenge Problem

2 Part 1: What is the benefit of numpy arrays?

Part 1a: Numpy arrays are efficient!

Say we want to convert a list of heights in inches to heights in centimeters

Uses the conversion 1 inch = 2.54 cm

```
[5]: height_in = [60, 72, 65, 64]
     unit_conv = 2.54
```

Wouldn't it be nice if there was a simple command to multiply each element of our list, height_in, by our converstion factor, 2.54.

Try running this line of code to see if it works.

```
[12]: height_cm = height_in * unit_conv
```

```
             ␣
 ↪---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call␣
 ↪last)

        <ipython-input-12-df031e499546> in <module>()
    ----> 1 height_cm = height_in * unit_conv


        TypeError: can't multiply sequence by non-int of type 'float'
```

As you saw, you got an error 'can't multiply sequence by non-int of type 'float"

Understand that error by using the FUNCTIONS print() and type() to print the object types of height_in and unit_conv.

[6]:
```
print(type(height_in))
print(type(unit_conv))
```

```
<class 'list'>
<class 'float'>
```

So we'll have to try another strategy to create a list of heights in cm, height_cm.

Intialize a list called height_cm.

Use the METHOD (a FUNCTION that "belongs" to an object) .append() to convert each element of height_in to a measurement of height in cm.

[11]:
```
height_cm = list()

height_cm.append(height_in[0] * 2.54)
height_cm.append(height_in[1] * 2.54)
height_cm.append(height_in[2] * 2.54)
height_cm.append(height_in[3] * 2.54)

print(height_cm)
```

```
[152.4, 182.88, 165.1, 162.56]
```

That was a lot of lines of code to do a simple task! We'll see in future lessons that we could automate this process (of typing each line manually) using LOOPS.

But there is actually an even easier (and more efficient) way to do this using the PACKAGE, numpy.

Import the PACKAGE numpy.

You can import numpy with the command: import numpy Then you would have to type out numpy.array, for example to create a numpy array

But you can change how you call numpy by importing NUMPY as a name with the command: import numpy as NAME (typically the name you'll see is np)

But just a note you can import numpy as ANY name…even your own name, but that wouldn't be that logical or efficient.

[ ]:
```
import numpy as np
```

Convert your list, height_in to a numpy ARRAY.

[ ]:
```
height_np = np.array(height_in)
```

Now we can do that simple command we wanted to do before with our list but could not.

[ ]:
```
height_np_cm = height_np * unit_conv
```

[ ]:
```
print(height_np_cm)
```

Part 1b: Numpy arrays are faster!

Say we wanted to add two long lists of numbers together. It would take much less time if we used numpy arrays instead of lists! The following block of code will show you just that– numpy arrays allow for more efficient calculations than lists.

Note that some of the concepts in this block of code you haven't seen before. Try to understand what is going and then run the code!

[16]:
```
import time
import numpy as np
SIZE = 10000000

#REGULAR PYTHON LISTS, L1 and L2
#range(SIZE) makes a list of numbers from 1 to the variable SIZE, which equals␣
 ↪10000000
L1 = range(SIZE)
L2 = range(SIZE)

#NUMPY ARRAYS, N1 and N2
#np.arange(SIZE) makes a numpy array of numbers from 1 to the variable SIZE,␣
 ↪which equals 10000000
N1 = np.arange(SIZE)
N2 = np.arange(SIZE)

#Record the start time in the variable, start
start = time.time()
#add the two python lists, L1 and L2 using loops and store in result
result = [x+y for x,y in zip(L1,L2)]
#print how long the addition took!
print("python list took", time.time()-start)

#Record the start time in the variable, start
start = time.time()
#add the two numpy arrays, N1 and N2 and store in result
result = N1+N2
#print how long the addition took!
print("numpy list took",time.time()-start)
```

```
python list took 1.0505850315093994
numpy list took 0.2186439037322998
```

3 PART 2: Let's work with numpy arrays, functions, and methods!

```
[29]: my_array = np.array([[100, 125, 115, 104], [1.0, 3.4, 2.7, 5.5], [15,  15,  24,␣
       ↪ 5  ]])
```

Find the max in the first list using the METHOD np.max(). Store answer in the VARIABLE max_first and print with the FUNCTION print.

```
[31]: max_first  = np.max(my_array[0])
      print(max_first)
```

```
125.0
```

Find the min in the second list using the METHOD np.min(). Store answer in the VARIABLE min_second and print with the FUNCTION print.

```
[32]: min_second = np.min(my_array[1])
      print(min_second)
```

```
1.0
```

Use the METHOD np.sort() to sort the third list from smallest to largest. Store in the VARIABLE my_array3_sorted and print with the FUNCTION print.

```
[37]: my_array3_sorted = np.sort(my_array[2])
      print(my_array3_sorted)
```

```
[  5.  15.  15.  24.]
```