Noah Anderson
Module 1
6/6/20224

Question 1

a.

Shown below is the R code output:

```
listed_range <- c(265, 270, 272, 223)
tested_range <- listed_range
tested_range[2] <- 289
listed_range
```

[1] 265 270 272 223

```
tested_range
```

[1] 265 289 272 223

b.

In [196…

```python
# Define range as a list
listed_range = [265, 270, 272, 223]

# Create tested_range
tested_range = listed_range

# Change the first element
tested_range[1] = 289

# Print the lists
print(listed_range)
print(tested_range)
```

```
[265, 289, 272, 223]
[265, 289, 272, 223]
```

In R, `listed_range` and `tested_range` differ in the second element (2nd according to R indexing). In Python, the lists are identical. This is because in R, vectors "copy" on modify, meaning that when you change something in the second vector it creates a copy and keeps the original as it was. In Python, lists are "modified in place", meaning a copy

is not made when you make changes to the second defined list. Instead the `tested_range` and `listed_range` refer to the same list.

c.

In [162…
```
# Define range as a tuple
listed_range = (265, 270, 272, 223)

# Define tested_range
tested_range = listed_range

# Redefine the 1st element
tested_range[1] = 289

# Print tuples
print(listed_range)
print(tested_range)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[162], line 3
      1 listed_range = (265, 270, 272, 223)
      2 tested_range = listed_range
----> 3 tested_range[1] = 289
      4 print(listed_range)
      5 print(tested_range)

TypeError: 'tuple' object does not support item assignment
```

The error "'tuple' object does not support item assignment" occurs because tuples are immutable meaning that once a tuple has been defined, you cannot make changes to its elements.

d.

In [166…
```
import array as arr

# Define range as an array using the array package
listed_range = arr.array('i', [265, 270, 272, 223])
tested_range = listed_range
tested_range[1] = 289

print(listed_range)
print(tested_range)
```

```
array('i', [265, 289, 272, 223])
array('i', [265, 289, 272, 223])
```

Arrays also modify in place like lists.

e.

In [170…
```
import numpy as np

# Define range as a NumPy array
listed_range = np.array([265, 270, 272, 223])
tested_range = listed_range
tested_range[1] = 289

print(listed_range)
print(tested_range)
```

```
[265 289 272 223]
[265 289 272 223]
```

NumPy arrays also modify in place.

f.

In [198…
```
listed_range = np.array([265, 270, 272, 223])

# Create a copy of listed_Range
tested_range = listed_range.copy()

# Change the elment on the copy
tested_range[1] = 289

# Define safe range array as 90% of tested_range
safe_range = tested_range * .9

print(listed_range)
print(tested_range)
print(safe_range)
```

```
[265 270 272 223]
[265 289 272 223]
[238.5 260.1 244.8 200.7]
```

The `copy` function allows one to "modify on copy".

Question 2

In [178…
```
class_list = ["SUV", "Sedan", "Sedan", "SUV"]
class_tuple = ("SUV", "Sedan", "Sedan", "SUV")
class_array = np.array(["SUV", "Sedan", "Sedan", "SUV"])
class_set = {"SUV", "Sedan", "Sedan", "SUV"}
class_dict = {
    'Class' : ["SUV", "Sedan", "Sedan", "SUV"]
}
```

```
print(class_list)
print(class_tuple)
print(class_array)
print(class_set)
print(class_dict)
```

```
['SUV', 'Sedan', 'Sedan', 'SUV']
('SUV', 'Sedan', 'Sedan', 'SUV')
['SUV' 'Sedan' 'Sedan' 'SUV']
{'SUV', 'Sedan'}
{'Class': ['SUV', 'Sedan', 'Sedan', 'SUV']}
```

A list, dictionary, tuple, array, and set could all be used to store this information, but not all are approriate. The least appropriate of these four would be the set since it takes only the unique values. Tuples may not be appororiate either depending on if you need to change any of the elements. A NumPy array, dictionary, and a list seem the most appropriate since they are both mutable.

Question 3

In [202…

```
# Explicitly redifining listed_range since the name has been used many times
listed_range = np.array([265, 270, 272, 223])

# Create for loop that iterates over length of the listed_range
for i in range(len(listed_range)):

    if listed_range[i] >= 270:

        # print the i'th element of range and class
        # if the listed range is over 270
        print(listed_range[i], class_list[i])
```

```
270 Sedan
272 Sedan
```

Question 4

In [185…

```
listed_range_np_copy = np.copy(listed_range)
listed_range_base_copy = listed_range.copy()

listed_range_np_copy[1] = 289
listed_range_base_copy[1] = 289

print("Original:", listed_range)
print("NumPy Copy:", listed_range_np_copy)
print("Base Copy:", listed_range_base_copy)
```

```
Original: [265 270 272 223]
NumPy Copy: [265 289 272 223]
Base Copy: [265 289 272 223]
```

Both copies are sucusfully altered without the orignial being changed.

Question 5

In [204…

```python
import pandas as pd

# Define car dictionary
car_dict = {
    'Make' : ['Audi', 'Polestar', 'Tesla', 'Volvo'],
    'Model' : ['Q4 e-tron', '2', 'Model 3', 'XC40 Recharge'],
    'Class' : ['SUV', 'Sedan', 'Sedan', 'SUV'],
    'Price' : [43900, 48400, 46990, 53550],
    'Range' : [265, 270, 272, 223]
}

# Convert dictionary to a data frame with pandas
car_df = pd.DataFrame(car_dict)
print(car_df)
```

```
      Make          Model  Class  Price  Range
0     Audi       Q4 e-tron    SUV  43900    265
1  Polestar              2  Sedan  48400    270
2    Tesla        Model 3  Sedan  46990    272
3    Volvo  XC40 Recharge    SUV  53550    223
```

In [ ]: