Python for Selenium

Tuples

- Tuples are very similar to list but once a tuple is created, you cannot add, delete, replace, reorder elements.
- Tuples are immutable.

Creating Tuples

```
t1 = ()   # creates an empty tuple with no data
t2 = (11, 22, 33)
t3 = tuple([1, 2, 3, 4, 4])   # tuple from array
t4 = tuple("abc")   # tuple from string

print(t1)
print(t2)
print(t3)
print(t4)
```

Tuples functions

- Functions like max , min , len , sum can also be used with tuples.

```
t1 = (1, 12, 55, 12, 81)

print(min(t1))  #1
print(max(t1))  #  81
print(sum(t1))  #161
print(len(t1))  #5
```

Iterating through tuples

- Tuples are iterable using for loop

```
t = (11,22,33,44,55)

for i in t:
    print(i, end=" ") #11 22 33 44 55
```

Slicing tuples

- Slicing operators works same in tuples as in list and string.

```
t = (11,22,33,44,55)
print(t[0:2]) #(11,22)
print(t[2:4]) #(11,22)
```

List Vs Dictionary Vs Tuple

Lists	Dictionaries	Tuples

List=[10,12,14]	Dict={**"John"**:26, **"Mary"**: 30}	Tup1=(**"10,20,30"**) or
		Tup2=10,20,30
		
		Tup3=(**"John"**,**"Scott"**) or
		Tup4=**"John"**,**"Scott"**

print(List[0])	print(Dict[**"Mary"**])	print(Tup1[0])
Allow duplicates	Duplicates Keys NOT allowed but duplicate values allowed.	Allow duplicates. Faster than Lists.
List[0]=100	Dict[**"John"**]=35	Tuple1[0]=100 #*Type error*
Mutable	Mutable	Immutable – Values can't changed once assigned
[ ]	{ }	( )

Slicing can be done.
List=[10,20,30]
print(List[1:2]) #*[20]*
	Slicing can't be done.	Slicing can be done.
		tup=(10,20,30,40,50)
		print(tup[1:4]) #*(20, 30, 40)*

Usage: use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	When you need a logical association between Key:value pair. When you need fast lookup for your data based on a custom key.	Use tuples when your data cannot change. A tuple is used in combination with a dictionary, for example a tuple might represent a key, because its immutable.

List Vs Tuple

```python
a = ["apples", "bananas", "oranges"]
a[0] = "berries"
print (a) #['berries', 'bananas', 'oranges'] # Perfect! the first item of a has changed.
```

```python
a = ("apples", "bananas", "oranges")
a[0] = "berries"    #We get an error saying that a tuple object doesn’t support item assignment.
```

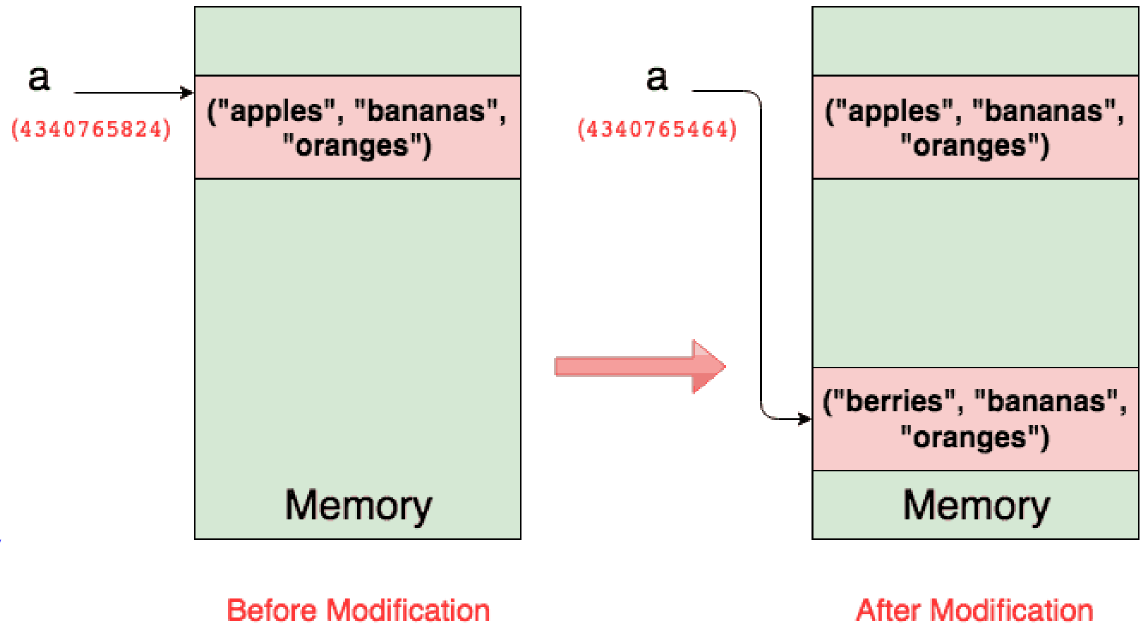
#The reason we get this error is because tuple objects are immutable which means you can’t modify a tuple object after it’s created.

List Vs Tuple…

```
a = ("apples", "bananas", "oranges")
a = ("berries", "bananas", "oranges")

print(a) # ('berries', 'bananas', 'oranges')
```

#Are we actually modifying the first item in tuple a with the code above?
#The answer is No, absolutely not.


a
(4340765824)
("apples", "bananas", "oranges")
Memory
Before Modification
a
(4340765464)
("apples", "bananas", "oranges")
("berries", "bananas", "oranges")
Memory
After Modification

https://www.afternerd.com/blog/difference-between-list-tuple/

File Handling

- We can use File handling to read and write data to and from the file.
- File Operations
  - Opening a File
  - Closing a file
  - Writing data in to file
  - Reading data from a file
  - Appending data
  - Looping through the data using for loop

Opening & Closing Files

- Before reading/writing you first need to open the file. Syntax of opening a file is.

  ```
  f = open(filename, mode)
  ```

- After you have finished reading/writing to the file you need to close the file using close() method like this,

  ```
  f.close()  # where f is a file pointer
  ```

- **Different modes of opening a file are**

MODES	**DESCRIPTION**
"r"	Open a file for read only
"w"	Open a file for writing. If file already exists its data will be cleared before opening. Otherwise new file will be created
"a"	Opens a file in append mode i.e to write a data to the end of the file

Writing Data into File

```
f = open('C:\DemoFiles\myfile.txt', 'w')     # open file for writing
f.write('this first line\n')    # write a line to the file
f.write('this second line\n')   # write one more line to the file
f.close()                       # close the file
```

Reading data from the file

- To read data back from the file you need one of these three methods.

read([number])	Return specified number of characters from the file. if omitted it will read the entire contents of the file.
readline()	
readlines()	Read all the lines as a list of strings in the file

Reading Data from File

```
#Reading all the data at once.
f = open('C:\DemoFiles\myfile.txt', 'r')
print(f.read()) # read entire content of file at once
f.close()
```

```
#Reading all lines as an array.
f = open('C:\DemoFiles\myfile.txt', 'r')
print(f.readlines()) # read entire content of file at once
f.close
```

```
#Reading only one line.
f = open('C:\DemoFiles\myfile.txt', 'r')
print(f.readline()) # read the first line
f.close()
```

Appending data

- To append the data you need to open the file in *'a'* mode.

```
#Appending data

f = open('C:\DemoFiles\myfile.txt', 'a')
f.write("this is third line\n")
f.close()
```

Looping through the data using for loop

```
#Looping through the data using for loop
f = open('C:\DemoFiles\myfile.txt', 'r')
for line in f:
    print(line)
f.close()
```