Python – Lists and tuples

List

- Unlike arrays, Lists are used to store different types of data

```
example engineer = [10,"Otto", "Mocheko", "Nathaniel", 50, 72,
‘M’]
         empty_list = [] # Empty list
```

- Indexing and Slicing can be applied on a list

```
example print(engineer[1]) # Gives "Otto"

         print(engineer[0: 3: 1]) # Prints [10, "Otto", "Mocheko"]

         engineer[::] # Prints all elements
```

Examples

- Create a list with numbers

```
numbers = [5, 10, 15, 20, 25]
print(numbers)
print("numbers[0]: {} numbers[2]: {}".format(numbers[0], numbers[2]))
```

- Create a list with strings

```
names = ["Elijah", "Osei", "Olamide"]
print(names)
print("names[0]: {} namdes[2]: {}".format(names[0], names[2]))
```

- Create a list with different data types

```
diff_types = [10, "Otto", "Mocheko", "Nathaniel", 50, 72, 89]
print(diff_types)
print ...
```

Creating list with range()

```
>>> num = list(range(4, 9, 2))
>>> print(num)
[4, 6, 8]
```

Updating list

- Creation

```
>>> lst = list(range(1, 5))
>>> print(lst)
[1, 2, 3, 4]
```

- Append

```
>>> lst.append(9)
>>> print(lst)
[1, 2, 3, 4, 9]
```

Updating List

- Update - 1

```
>>> lst[1] = 8
>>> print(lst)
[1, 8, 3, 4, 9]
```

- Update - 2

```
>>> lst[1: 3] = 10, 11
>>> print(lst)
[1, 10, 11, 4, 9]
```

Updating a list

- Delete

```
>>> del lst[1]
>>> print(lst)
[1, 11, 4, 9]
```

- Remove

```
>>> lst.remove(11)
>>> print(lst)
[1, 4, 9]
```

- Reverse

```
>>> lst.reverse()
>>> lst
[9, 4, 1]
```

Concatenation of two lists

- “+” Operator is used to join two lists

```
>>> lst1 = [5, 10, 15]
>>> lst2 = [20, 25]
>>> print(lst1 + lst2)
[5, 10, 15, 20, 25]
```

Repetition of List

- “*” is used to repeat the list ‘n’ times

```
>>> nums = [10, 20, 30]
>>> print(nums * 2)
[10, 20, 30, 10, 20, 30]
```

Membership of a list

- “In” and “not in” operators are used to check, whether an element belongs to the list or not

```
x = [1, 2, 3, 4, 5, 6]
a = 2
print(a in x) # Returns True, if the item is found in the List

x = [1, 2, 3, 4, 5, 6]
a = 7
print(a not in x) # Returns True, if the item is not found in
the list
```

Aliasing and cloning Lists

- Giving a new name for the existing list
  - This method does not copy the list but rather copies the reference of the list to the second variable.

```
x = [20, 30, 40]
y = x # In this case, No separate memory will be allocated for y
```

- Cloning / Making a copy

```
>>> x = [10, 20, 30, 40, 50]
>>> y = x[:]
>>> x[1] = 100
>>> print(x)
[10, 100, 30, 40, 50]
>>> print(y)
[10, 20, 30, 40, 50]
# Note: Changes made in one list will not reflect on the other
```

Aliasing and cloning Lists

- Copying a list will only allow primitive values in the list to be copied.
- To copy a nested list, a different approach must be used.
  - One is to use the copy module.

  -
    ```
    >>> old_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    >>> new_list = old_list.copy()
    >>> old_list.append([4, 4, 4])
    >>> print("New list:", new_list)
    New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    >>> print("Old list:", old_list)
    Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9], [4, 4, 4]]
    ```
  - As you can see, the old value being changed does not affect the newly copied list.

To find the common items

```
# To find the common item given two lists

lst = ["Alex", "Abdul", "Phillip", "Janet"]
lst2 = ["Amos", "Phillip", "Leonex", "Fazul"]

# Convert them into sets
lst = set(lst)
lst2 = set(lst2)

# Filter intersection of two sets
lst3 = lst.intersection(lst2)

# Convert back into the list
common = list(lst3)

print(common)
```

Nested List

```
# To create a list with another list as element

lst = [5, 10, 15, [40, 50]]
print(lst)
```

List Comprehensions

- Creation of new list from an iterable object (set tuple, dictionary, list or range) that satisfies a given condition

```
# Create a list with squares of integers from 1 to 5

# Method 1
squares = []
for i in range(1, 6):
   squares.append(i ** 2)
print(squares)

# Method 2
squares = []
squares = [i ** 2 for i in range(1, 6)]
print(squares)
```

List Comprehensions

```
# Get even squares from 1 to 10

even_squares = [i ** 2 for i in range(1, 11) if i % 2 == 0]
print(even_squares)
```

List Comprehensions

```
# Adding the elements of two list one by one

x = [10, 20, 30]
y = [1, 2, 3, 4]

lst = []

# Method 1
for i in x:
 for j in y:
   lst.append(i + j)


# Method 2
lst = [i + j for i in x for j in y]


# concatenate two lists
lst = [i + j for i in "ABC" for j in "DE"]
print(lst)
```

Exercise

- Reverse the elements of a list
- Find the minimum and maximum element in a list of elements
- Check how many times an element occurs in a list
- Create a languages list and search for a particular language

Tuple

- A tuple is similar to list but is immutable

Creating tuples

```
To create empty tuple
tup1 = ()

Tuple with one item
tup2 = (10, )

Tuple with different dtypes
tup3 = (10, 20, 2.3, "Jonnes", "M")

Tuple with no braces
tup4 = 5, 20, 35, 50

Create a tuple from the list
lst = [10, 20, 2.3, "Jonnes", "M"]
tup5 = tuple(lst)

Create tuple from range
tup6 = tuple(range(10))
```

Accessing Tuples

- Accessing items in the tuple can be done by indexing or slicing method, similar to that of list

Basic Operations On Tuples

```
s = (10, "Jonnes", 10, 20, 30, 40, 50)

To find the length of the tuple
print(len(s))

Repetition operator
mark = (25.000, ) * 4
print(mark) # (25.0, 25.0, 25.0, 25.0)

Concatenate the tuples using +
co = s + mark
print(co)  # (10, 'Jonnes', 10, 20, 30, 40, 50, 25.0, 25.0,
25.0, 25.0)
```

Functions To Process Tuples

- len() - len(tpl) - Returns the number of elements in the tuple
- max() - max(tpl) - Returns the biggest element in the tuple
- min() - min(tpl) - Returns the smallest element in a tuple
- count() - tpl.count(x) - Returns how many times the element 'x' is found in the tuple
- index() - tpl.index(x) - Returns the first occurence of the element x in tuple. If not found, Raises ValueError
- sorted() - sorted(tpl) - Sorts the elements of the tuple into ascending order sorted(tuple_name, reverse=True) will sort in reverse order

Exercise

- Find the occurrence of an element in a tuple
- Insert a new item into a tuple at a specified location
- Replace an existing element of a tuple with new element
- Delete an element from a particular position in the tuple

See you at the next session!