PYTHON TUTORIAL FOR BEGINNERS

Source: www.youtube.com/@RishabhMishraOfficial

Assignment - 06

10 Question on

- **List,**
- **Tuple,**
- **Set**
- **& Dictionary**

Top 10 Questions on Data Structures in Python

1. Find the Intersection (common elements) of Two Lists
2. Find the Most Frequent Element in a List
3. Find Cumulative Sum of a List.
4. Remove Duplicates from a List – 2 methods
5. Find the index of an element in a tuple.
6. Find the Most Frequent Value in a dictionary
7. Merge Dictionaries with Summation
8. Flatten a Nested Dictionary
9. Sort a Dictionary by Values
10. Access values from a nested dictionary

Lambda function in Python

A **lambda** function in Python is a small, anonymous function defined using the **lambda keyword**. It can have any number of arguments, but only **one** expression, which is evaluated and returned.

Lambda functions are often used for short, simple operations without needing to define a full function using def.

Syntax

```
lambda arguments: expression
```

- Arguments: Input to the function.
- Expression: A single statement or operation that the lambda function will return.

Lambda function examples

 # Add two numbers using a regular function:

```
def add(x, y):
    return x + y
print(add(3, 5))  # Output: 8
```

 # Add two numbers using a lambda function:

```
add = lambda x, y: x + y
print(add(3, 5))  # Output: 8
```

 # Custom Sorting - Sort a list of tuples by the second element:

```
data = [(1, 'b'), (3, 'a'), (2, 'c')]
sorted_data = sorted(data, key=lambda x: x[1])
print(sorted_data)
# Output: [(3, 'a'), (1, 'b'), (2, 'c')]
```

Q1 Find the Intersection (common elements) of Two Lists?

```
list1 = [1,2,4,5]
list2 = [4,5,6,7,8]

# using for loop
def intersection_loop(lst1, lst2):
    common_list = []
    for item in lst1:
```

```
        if item in lst2 and item not in common_list:
            common_list.append(item)
    return common_list

print(intersection_loop(list1, list2))

# using List comprehension
def intersection_comp(lst1, lst2):
    return [item for item in lst1 if item in lst2]

print(intersection_comp(list1, list2))
```

Q2 Find the Most Frequent Element in a List?

```
numbers = [1,2,2,3,3,3,4,7,7,7,7]

def most_freq(lst):
    max_count = 0
    most_freq = None
    for item in lst:
        count = lst.count(item)
        if count > max_count:
            max_count = count
            most_freq = item
    return most_freq

print(most_freq(numbers))
```

Q3 Find Cumulative Sum of a List

```
numbers = [1, 2, 3, 4]

def cumulative_sum(lst):
    cum_sum = []
    total = 0
    for num in lst:
        total += num
        cum_sum.append(total)
    return cum_sum

print(cumulative_sum(numbers))

# Using List Comp:
print([sum(numbers[:i + 1]) for i in range(len(numbers))])
```

Q4 Remove Duplicates from a List

```
fruits = ["apple", "banana", "mango", "apple", "banana"]

# using loop
def remove_duplicates(lst):
    unique = []
    seen = set()
    for item in lst:
        if item not in seen:
            unique.append(item)
            seen.add(item)
    return unique
```

```
print(remove_duplicates(fruits))

# without seen, but not good for large dataset -list
def remove_duplicates(lst):
    unique = []
    for item in lst:
        if item not in unique:
            unique.append(item)
    return unique

print(remove_duplicates(fruits))

# using set constructor
print(list(set(fruits)))
```

Q5 Find the index of an element in a tuple

```
my_tuple = (1, 10, 2, 3, 4)

def find_index(tup, elem):
    return tup.index(elem) if elem in tup else -1

print(find_index(my_tuple,100))
```

Q6 Find the Most Frequent Value in a dictionary

```
data = {'a': 1, 'b': 2, 'c': 1, 'd': 3, 'e': 1}
```

```
def most_freq(dct):
    frequency = {}
    for value in dct.values():
        if value not in frequency:
            frequency[value] = 0
        frequency[value] += 1 # 1:1, 2:1, 1:2,3:1, 1:3
    max_value = max(frequency, key=frequency.get)
    return max_value

print(most_freq(data))
```

Q7 Merge Dictionaries with Summation

```
dict1 = {'a': 10, 'b': 20, 'c': 30}
dict2 = {'b': 15, 'c': 35, 'd': 25}

def merge_dict(dict1, dict2):
    result = dict1.copy()
    for key, value in dict2.items():
        if key in result:
            result[key] += value
        else:
            result[key] = value
    return result

print(merge_dict(dict1, dict2))
```

Q8 Flatten a Nested Dictionary

```
data = {'a': {'b': {'c': 42}, 'd': 7}, 'e': 10}
#o/p {a.b.c: 42, a.d: 7, e: 10}


def flatten_dict(data, parent_key= '', sep = '.'):
    items = {} #initialize empty dict to store flattened items
    for key, value in data.items():
        # combine current key with parent key
        new_key = f"{parent_key}{sep}{key}" if parent_key else
key
        if isinstance(value, dict): # check if dict or not
            # recursive flatten the nested dict
            items.update(flatten_dict(value, new_key, sep))
        else:
            # adding key-value to flatten dict
            items[new_key] = value
    return items

print(flatten_dict(data))

data = [1,2,4]
print(isinstance(data, list))
```

Q9 Sort a Dictionary by Values

```
data = {'a': 5, 'b': 9, 'c': 2, 'd': 7}
```

```
def sort_by_values(data):
    sorted_items = sorted(data.items(),
                          key = lambda item: item[1],
                          reverse=True)
    return {key: value for key , value in sorted_items}

print(sort_by_values(data))

print(sorted([1,2,0,2,8], reverse=True))

print(data.items())
```

Q10 Access values from a nested dictionary

```
    data = {
        "level1": {
            "level2": {
                "level3": {
                    "value1": 10,
                    "value2": [1, 2, {"deep_key": 42}],
                    "value3": {"inner_key": "target"}
                },
                "other_key": 99
            },
            "list_key": [
                {"list_inner_key1": 88},
                {"list_inner_key2": {"deep_list_key": 77}}
            ]
        }
    }
```

```
# Tasks to Access Elements:
```

- Retrieve 42
- Retrieve "target"
- Retrieve 77

#10 Access values from a nested dictionary – Solution

Tasks to Access Elements

1. **Retrieve 42.**
   Path: data -> level1 -> level2 -> level3 -> value2 -> [2] -> deep_key

```
print(data["level1"]["level2"]["level3"]["value2"][2]["deep_key"])
```

2. **Retrieve "target".**
   Path: data -> level1 -> level2 -> level3 -> value3 -> inner_key

```
print(data["level1"]["level2"]["level3"]["value3"]["inner_key"])
```

3. **Retrieve 77.**
   Path: data -> level1 -> list_key -> [1] -> list_inner_key2 -> deep_list_key

```
print(data["level1"]["list_key"][1]["list_inner_key2"]["deep_list_key"])
```

Python Tutorial Playlist: **Click Here**

https://www.youtube.com/playlist?list=PLdOKnrf8EcP384Ilxra4UlK9BDJGwawg9