Python Sequence and Collections – Operations, Functions, Methods

Python Sequence

A sequence is a group of items with a deterministic ordering. The order in which we put them in is the order in which we get an item out from them.

Check the output given in the following examples and understand how it came…..

Eg:

1) String

```
>>> name=str()
>>> name
>>> type(name)
>>> name=str('Ayushi')
>>> name[3]
```

2) List

```
>>> groceries=['milk','bread','eggs']
>>> groceries[1]
```

3) Tuples

```
a. # alphabets tuple
b. alphabets = ('a', 'e', 'i', 'o', 'g', 'l', 'i', 'u')
c.
d. # index of 'i' in alphabets
e. index = alphabets.index('e')   # 2
f. print('The index of e:', index)
```

Sequence operations

Check the output given in the following examples and understand how it came…..

Eg:

1. **Concatenation**

```
>>> 'Ayu'+'shi'
```

2. **Integer Multiplication**

```
>>> 'ba'+'na'*2
```

3. **Membership**

```
>>> 'men' in 'Disappointment'
```

4. **Python Slice**

```
>>> 'Ayushi'[1:4]
```

Sequence functions

A function is a block of code to carry out a specific task, will contain its own scope and is called by name.
Eg: len(),min() and max()

Sequence methods

A method in python is somewhat similar to a function, except it is associated with object/classes.

Eg: index(), count()

Python Collection

Python collections, unlike a sequence, do not have a deterministic ordering.

Eg: include sets and dictionaries.

Questions:

1: Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).

```
n=int(input("Input a number "))
d = dict()

for x in range(1,n+1):
    d[x]=x*x
print(d)
```

2: Write a Python program to sum all the items in a dictionary

```
my_dict = {'data1':100,'data2':-54,'data3':247}
print(sum(my_dict.values()))
```

Comprehensions in Python

Comprehensions in Python provide us with a short and concise way to construct new sequences/collections.

List Comprehensions:

List Comprehensions provide an elegant way to create new lists. The following is the basic structure of a list comprehension:
output_list = [*output_exp* for *var* in *input_list* if (*var* satisfies this condition)]

List Comprehensions vs loops

The list comprehensions are **more efficient** both **computationally** and in terms of **coding space and time** than a for loop. Typically, they are written in a single line of code.

```
# Example for Constructing output list WITHOUT Using List comprehensions
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_list = []
for var in input_list:
    if var % 2 == 0:
        output_list.append(var)
print("Output List using for loop:", output_list)
```

```
# Example for Constructing output list Using List comprehensions

input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
list_using_comp = [var for var in input_list if var % 2 == 0]
print("Output List using list comprehensions:",  list_using_comp)
```

```
# Example for Constructing output list using for loop
output_list = []
for var in range(1, 10):
    output_list.append(var ** 2)
print("Output List using for loop:", output_list)
```

```
# Example for Constructing output list using list comprehension
list_using_comp = [var**2 for var in range(1, 10)]
print("Output List using list comprehension:",  list_using_comp)
```

```
# Another Example using list comprehension
a_list = [1, '4', 9, 'a', 0, 4]
```

```
squared_ints = [ e**2 for e in a_list if isinstance(e, int)]
print (squared_ints)
```

Dictionary Comprehensions:

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

```
output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}
```

```
# Example for Constructing output dictionary Using loops
input_list = [1, 2, 3, 4, 5, 6, 7]
output_dict = {}
for var in input_list:
    if var % 2 != 0:
        output_dict[var] = var**3
print("Output Dictionary using for loop:", output_dict )
```

```
# Example Using Dictionary comprehensions for constructing output dictionary
input_list = [1,2,3,4,5,6,7]
dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}
print("Output Dictionary using dictionary comprehensions:", dict_using_comp)
```

Set Comprehensions:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }.

```
# Example Using loops for constructing output set
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
output_set = set()
for var in input_list:
    if var % 2 == 0:
        output_set.add(var)
print("Output Set using for loop:", output_set)
```

```
# Example Using Set comprehensions for constructing output set
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
print("Output Set using set comprehensions:",set_using_comp)
```