data_structure

June 28, 2024

[4]:
```
#Q-1- Discuss string slicing and provide examples.

#Answer
#Slicing is a method of creating a sublist of consecutive elements drawn from another list.
#It follows the format string[start:end]

#Example
# with the string given if we want to slice out 'Hello'
my_string="Hello World"
my_string[0:5]
```

[4]:
```
'Hello'
```

[3]:
```
# if we want to slice out 'World'
my_string[6:13]
```

[3]:
```
'World'
```

[5]:
```
#another way of slicing
my_string[:13]#In this by default start point assumed is 0
```

[5]:
```
'Hello World'
```

[3]:
```
# Q-2- Explain the key features of lists in Python.

#Anwer
#The important characteristics of Python lists are as follows:
#1.Lists are ordered.
#2.Lists can contain any arbitrary objects.
#3.List elements can be accessed by index.
#4.Lists can be nested to arbitrary depth.
#5.Lists are mutable.
#6.Lists are dynamic.

#Example-
my_list=["ashu","Abhi","2+i3"]
```

```
#len(): Returns the number of elements in the list.
#sorted(): Returns a new sorted list of the elements in the original list.
#min(): Returns the smallest element in the list.
#max(): Returns the largest element in the list.
#sum(): Returns the sum of all elements in the list
```

[16]:
```
#len(): Returns the number of elements in the list.
#Examples
len(my_list)
my_list
```

[16]:
```
['ashu', 'Abhi', '2+i3', 'lucky', 'lucky', 'lucky', 'lucky', 'lucky']
```

[13]:
```
#to add some new element
my_list.append("lucky")
my_list
```

[13]:
```
['ashu', 'Abhi', '2+i3', 'lucky', 'lucky', 'lucky', 'lucky', 'lucky']
```

[30]:
```
#to add
a=[1,3,4,6,5,6]
sum(a)
```

[30]:
```
25
```

[31]:
```
#to sort
sorted(a)
```

[31]:
```
[1, 3, 4, 5, 6, 6]
```

[32]:
```
#Q-3-Describe how to access, modify and delete elements in a list with example.

#Answer

#Python Lists have various built-in methods to remove items from the list.␣
 ↪Apart from these, we can also use different methods to remove an element␣
 ↪from the list by specifying its position.

#1. Remove Elements from the List using remove()
lst = ['Iris', 'Orchids', 'Rose', 'Lavender',
    'Lily', 'Carnations']
print("Original List is :", lst)

# using remove()
lst.remove('Orchids')
```

```
print("After deleting the item :", lst)
```

```
Original List is : ['Iris', 'Orchids', 'Rose', 'Lavender', 'Lily', 'Carnations']
After deleting the item : ['Iris', 'Rose', 'Lavender', 'Lily', 'Carnations']
```

[33]:
```
#2. Remove Element from the List using pop()
lst = ['Iris', 'Orchids', 'Rose', 'Lavender',
    'Lily', 'Carnations']
print("Original List is :", lst)

# using pop() to delete item
# ('Orchids' at index 1) from the list
a = lst.pop(1)
print("Item popped :", a)
print("After deleting the item :", lst)
```

```
Original List is : ['Iris', 'Orchids', 'Rose', 'Lavender', 'Lily', 'Carnations']
Item popped : Orchids
After deleting the item : ['Iris', 'Rose', 'Lavender', 'Lily', 'Carnations']
```

[34]:
```
#3. Remove Element from the List using del()
lst = ['Iris', 'Orchids', 'Rose', 'Lavender',
    'Lily', 'Carnations']
print("Original List is :", lst)

# using del statement
# to delete item (Orchids at index 1)
# from the list
del lst[1]
print("After deleting the item :", lst)
```

```
Original List is : ['Iris', 'Orchids', 'Rose', 'Lavender', 'Lily', 'Carnations']
After deleting the item : ['Iris', 'Rose', 'Lavender', 'Lily', 'Carnations']
```

[36]:
```
#4. To access and replace items

a=lst[1]#index acces
a
```

[36]:
```
'Rose'
```

[38]:
```
lst[1]="mango"#replace
lst
```

[38]:
```
['Iris', 'mango', 'Lavender', 'Lily', 'Carnations']
```

[39]:
```
# Q- 4- Compare and contrast tuples and lists with examples

# Answer

#*Lists are delineated using square brackets, like so: [1, 2, 3]. Tuples, in
 ↪contrast, are encased in parentheses: (1, 2, 3)
#*Both are used to store the data, and the values stored can be accessed using
 ↪indexes
#*The key difference between both is that lists are mutable, and tuples are
 ↪immutable.

# Examples
# list
a=["a","b","c","d","e","f"]
a
```

[39]:
```
['a', 'b', 'c', 'd', 'e', 'f']
```

[40]:
```
#mutable
a[1]="n"
a
```

[40]:
```
['a', 'n', 'c', 'd', 'e', 'f']
```

[41]:
```
#Tuples
a=("a",2,3,5)
a
```

[41]:
```
('a', 2, 3, 5)
```

[42]:
```
#Immutable
a[1]=4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[42], line 2
      1 #Immutable
----> 2 a[1]=4

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

[45]:
```
# Q- 5-  Describe the key features of sets and provide examples of their use.

#Answer

#-Sets are unordered.
```

```
#-Set elements are unique. Duplicate elements are not allowed.
#-A set itself may be modified, but the elements contained in the set must be␣
 ↪of an immutable type.
#-It is a collection that is written with curly brackets and is both unindexed␣
 ↪and unordered.

#Reprsesantation
a={1,2,3,4,5,5,5,6,6,7}
a
```

[45]:
```
{1, 2, 3, 4, 5, 6, 7}
```

[46]:
```
set1 = {1,2,3,4,5,6}

print("Initial set")
print(set1)

# This method will remove
# all the elements of the set
set1.clear()

print("\nSet after using clear() function")
print(set1)
```

```
Initial set
{1, 2, 3, 4, 5, 6}

Set after using clear() function
set()
```

[48]:
```
#type casting

b=set([1,2,3,4])
b
```

[48]:
```
{1, 2, 3, 4}
```

[50]:
```
#to add element in set
b.add("d")
print(b)
```

```
{1, 2, 3, 4, 'd'}
```

[1]:
```
# Q- 6- Discuss the use cases of tuples and sets in Python programming.

#Answer
```

```
#Use cases of tuples
#Use of tuple where you cannot modify tuple

# Example 1 Employe Id assigned cannot changed
Emp_ID = (101,102,103,104,105,106)
Emp_ID[1]=109
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 10
      1 # Q- 6- Discuss the use cases of tuples and sets in Python programming.
      2
      3 #Answer
   (…)
      7
      8 # Example 1 Employe Id assigned cannot changed
      9 Emp_ID = (101,102,103,104,105,106)
---> 10 Emp_ID[1]=109

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

[3]:
```
#Example to count
Emp_ID = (101,102,103,104,105,106)
Emp_ID.count(101)
```

[3]: 1

[4]:
```
#to find position of element
tuple1= (1,2,3,4,5)
tuple1.index(3)
```

[4]: 2

[5]:
```
#to find Maximum and Minimum
tuple1=(102,105,109,10001)
max(tuple1)
```

[5]: 10001

[7]:
```
min(tuple1)
```

[7]: 102

[9]:
```
#Use case of set

#typecasting of list into set
```

```
list1=["Banana","mango","potato","book", "Banana","mango"]
set1=set(list1)
set1
```

[9]: {'Banana', 'book', 'mango', 'potato'}

```
[10]: #To add any item in set
set1.add(3)
set1
```

[10]: {3, 'Banana', 'book', 'mango', 'potato'}

```
[11]: #to remove any element
set1.remove(3)
set1
```

[11]: {'Banana', 'book', 'mango', 'potato'}

```
[12]: #to clear the set
set1.clear()
set1
```

[12]: set()

```
[14]: #Operations of set

s1 ={"Maths","Physics","Chemistry","English"}
s2={"Biology","Physics","Chemistry","Hindi"}
s1
```

[14]: {'Chemistry', 'English', 'Maths', 'Physics'}

```
[15]: s2
```

[15]: {'Biology', 'Chemistry', 'Hindi', 'Physics'}

```
[16]: #Union of sets
s1|s2
```

[16]: {'Biology', 'Chemistry', 'English', 'Hindi', 'Maths', 'Physics'}

```
[17]: s1.union(s2)
```

[17]: {'Biology', 'Chemistry', 'English', 'Hindi', 'Maths', 'Physics'}

```
[18]: #Intersection of sets
s1&s2
```

[18]:
```
{'Chemistry', 'Physics'}
```

[19]:
```
s1.intersection(s2)
```

[19]:
```
{'Chemistry', 'Physics'}
```

[20]:
```
#Difference of sets
s1-s2
```

[20]:
```
{'English', 'Maths'}
```

[22]:
```
#Symmetric difference
s1^s2
```

[22]:
```
{'Biology', 'English', 'Hindi', 'Maths'}
```

[24]:
```
#frozen set where we cannot add or remove anything

my_set=frozenset([1,2,3,3,3,2,5,6,7,1])
my_set
```

[24]:
```
frozenset({1, 2, 3, 5, 6, 7})
```

[25]:
```
my_set.add(5)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[25], line 1
----> 1 my_set.add(5)

AttributeError: 'frozenset' object has no attribute 'add'
```

[30]:
```
# Q- 7-- Describe how to add, modify and delete items in a dictionary with␣
 ↪examples

#Answer

# Dictionary
d={}
type(d)
```

[30]:
```
dict
```

[33]:
```
#To Access elements

#Examples
```

```
d= { "name": "Ashutosh","Email": "ashutosh.mishra@gmail.com","DOB": 1998 }
d
```

[33]: {'name': 'Ashutosh', 'Email': 'ashutosh.mishra@gmail.com', 'DOB': 1998}

[34]:
```
d["name"]
```

[34]: 'Ashutosh'

[35]:
```
d["Email"]
```

[35]: 'ashutosh.mishra@gmail.com'

[36]:
```
d["DOB"]
```

[36]: 1998

[37]:
```
#to Modify>>want to change name

d= { "name": "Ashutosh","Email": "ashutosh.mishra@gmail.com","DOB": 1998 }
d
```

[37]: {'name': 'Ashutosh', 'Email': 'ashutosh.mishra@gmail.com', 'DOB': 1998}

[38]:
```
d["name"]="Abhishek"
```

[39]:
```
d
```

[39]: {'name': 'Abhishek', 'Email': 'ashutosh.mishra@gmail.com', 'DOB': 1998}

[47]:
```
# to add Element can be added one at a time<<Address

d["Address"]="Sector 135 oida"
d
```

[47]:
```
{'name': 'Abhishek',
 'Email': 'ashutosh.mishra@gmail.com',
 'DOB': 1998,
 'Address': 'Sector 135 oida'}
```

[48]:
```
# to delete element

del d["Address"]
```

[49]:
```
d
```

[49]: {'name': 'Abhishek', 'Email': 'ashutosh.mishra@gmail.com', 'DOB': 1998}

[52]:
```
del d
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[52], line 1
----> 1 del d

NameError: name 'd' is not defined
```

[51]:
```
d
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[51], line 1
----> 1 d

NameError: name 'd' is not defined
```

[53]:
```
d= { "name": "Ashutosh","Email": "ashutosh.mishra@gmail.com","DOB": 1998 }
d.clear()
```

[54]:
```
d
```

[54]:
```
{}
```

[57]:
```
#to access keys only
d= { "name": "Ashutosh","Email": "ashutosh.mishra@gmail.com","DOB": 1998 }
d.keys()
```

[57]:
```
dict_keys(['name', 'Email', 'DOB'])
```

[59]:
```
d.values()
```

[59]:
```
dict_values(['Ashutosh', 'ashutosh.mishra@gmail.com', 1998])
```

[60]:
```
d.get("name")
```

[60]:
```
'Ashutosh'
```

[1]:
```
# Q -8-  Discuss the importance of dictionary keys being immutable and provide␣
 ↪examples

#Answer
```

```
#Immutability is essential for keys because it ensures that the dictionary can␣
 ↪efficiently look up values based on their keys. If a key was mutable, its␣
 ↪hash value could change, making it impossible to find the associated value␣
 ↪in the dictionary.
#Example use list , tuple , string to see immutability

d={[1,2,3,4,5]:"ABC"}#list
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 8
      1 # Q -8-  Discuss the importance of dictionary keys being immutable and␣
 ↪provide examples
      2
      3 #Answer
      4
      5 #Immutability is essential for keys because it ensures that the␣
 ↪dictionary can efficiently look up values based on their keys. If a key was␣
 ↪mutable, its hash value could change, making it impossible to find the␣
 ↪associated value in the dictionary.
      6 #Example use list , tuple , string to see immutability
----> 8 d={[1,2,3,4,5]:"ABC"}

TypeError: unhashable type: 'list'
```

[2]:
```
d={{1,2,3}:"abs"}#set
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 d={{1,2,3}:"abs"}#set

TypeError: unhashable type: 'set'
```

[4]:
```
d={(1,2,3):"abc"}#tuple
d
```

[4]:
```
{(1, 2, 3): 'abc'}
```

[ ]:
```
**************************End of Assignment**********************************
```