data_structure

August 14, 2024

Q1)Discuss string slicing and provide examples.

ANS:string is a set of characters.They are collection of ordered elements has specific index allocated to each character within the string. string slicing is the art of extracting specific portions(substrings) from a string by specifying start and end of indices. here,start index is included and end index is excluded in sliced string,also step index (optional) specifies gap between each index for slicing(skips the element). simple syntax: string[start:end]

[7]:
```
#example on slicing of string
string = "python"
string[1:4] # slice from index 1(included) to index 4(excluded)
```

[7]:
```
'yth'
```

[8]:
```
string[0:5:2] #string[start:end:step]
```

[8]:
```
'pto'
```

[9]:
```
string[0:]
```

[9]:
```
'python'
```

[25]:
```
string[-2]
```

[25]:
```
'o'
```

[28]:
```
string[-3:] #indicates include last three characters
```

[28]:
```
'hon'
```

[10]:
```
string[:-2] #indicates skip last two characters
```

[10]:
```
'pyth'
```

Q2)Explain key features of lists in python.

ANS:List:list is an in-build datatype an ordered collection of elements/objects in single variable with some index acquired by each element. They are created using [] brackets.

[10]:
```
#ex of list
l=[1,2,3,4,5]
l
```

[10]:
```
[1, 2, 3, 4, 5]
```

key features of lists in python are as follows:

1)Lists are ordered. It is collection of ordered elements.hence,maintains order of elements,allows you to access elements by their index. The order is preserved even after insertion and deletion operations.

[7]:
```
#ex1)two variables having same elements but in different order are not same
a=["Mango","apple","orange","banana"]
b=["orange","mango","banana","apple"]
a==b
```

[7]:
```
False
```

2)lists are heterogeneous. list can contain anything and everything.hence they are heterogeneous because elements in list could have datatypes like characters,strings,integer,boolean,complex number,float or may be combination of all datatypes, same datatype could appear multiple times(duplicate elements) in a single list or variable.even other list within a list is possible.They can be iterated by loops. They are also homogeneous as we can use one specific datatype in a single list.ex int

[11]:
```
#ex2)
l=[1,2,3,4] #only integer type in a list(homogeneous)
l
```

[11]:
```
[1, 2, 3, 4]
```

[12]:
```
type(l)
```

[12]:
```
list
```

[13]:
```
l=["jay","yash","anurag"] #only strings in a  list(homo)
l
```

[13]:
```
['jay', 'yash', 'anurag']
```

[14]:
```
l=['a','b','c'] #only characters in a list(homo)
l
```

[14]:
```
['a', 'b', 'c']
```

[15]:
```
l=["dhanashree",2.5,200,2+3j,True] #combination of different␣
 ↪datatypes(heterogeneous)
```

```
l
```

[15]:
```
['dhanashree', 2.5, 200, (2+3j), True]
```

[16]:
```
l=["uuu",100,"lll",100] #same datatype(value) appeared 2 times(duplicate elements allowed)
l
```

[16]:
```
['uuu', 100, 'lll', 100]
```

[111]:
```
#nesting of list:list that appears as element in another list
mylist=[[1,2,3],["joe","moe"]]
mylist
```

[111]:
```
[[1, 2, 3], ['joe', 'moe']]
```

3)list elements can be accessed by index individual elements in a list can be accessed using index in square brackets.starts from 0,1,2...thus,eack element has unique index corrosponding to its position in the list. Negetive indexing of elements can be accessed from end of the list.

[17]:
```
#ex3)
l=[12,"ions",3.4]
l[0] #0th index value
```

[17]:
```
12
```

[18]:
```
l[-1] #negetive index value
```

[18]:
```
3.4
```

[79]:
```
#deep copy:change in one variable will not reflect in another variable
kirti=["a","b","c"]
ajay=kirti.copy()
ajay
```

[79]:
```
['a', 'b', 'c']
```

[80]:
```
kirti
```

[80]:
```
['a', 'b', 'c']
```

[81]:
```
kirti[0]="d"
kirti
```

[81]:
```
['d', 'b', 'c']
```

[82]:
```
ajay #change in kriti not reflected in ajay
```

[82]:
```
['a', 'b', 'c']
```

[83]:
```
#shallow copy:change in one variable will reflect in another variable
kirti=[1,2,3,4]
ajay=kirti
kirti
```

[83]:
```
[1, 2, 3, 4]
```

[84]:
```
ajay
```

[84]:
```
[1, 2, 3, 4]
```

[85]:
```
kirti[0]=9
kirti
```

[85]:
```
[9, 2, 3, 4]
```

[86]:
```
ajay #kirti's values reflected in ajay
```

[86]:
```
[9, 2, 3, 4]
```

[88]:
```
#membership operators(in and not in)
l=["alooo","leleo",56]
l
```

[88]:
```
['alooo', 'leleo', 56]
```

[89]:
```
"leleo" in l
```

[89]:
```
True
```

[90]:
```
"leleo" not in l
```

[90]:
```
False
```

[91]:
```
#concatenation(+):creates new list by adding another list to it.
a=["you are great","you are clever"]
b=["i am the best"]
a+b
```

[91]:
```
['you are great', 'you are clever', 'i am the best']
```

[92]:
```
#replication(*):broadcast same msg multiple times
a*2
```

[92]:
```
['you are great', 'you are clever', 'you are great', 'you are clever']
```

[93]:
```
#len(),min(),max()
a=[1,2,3,4,5]
len(a)
```

[93]: 5

[94]:
```
min(a)
```

[94]: 1

[95]:
```
max(a)
```

[95]: 5

#list supports slicing It allows you to extract from both positive and negetive indices a specific portion(substrings) from the list by specifying start and end of the extracting portion.

[28]:
```
#ex4)
l=[1,2,"kpop",23.4,]
l
```

[28]: [1, 2, 'kpop', 23.4]

[29]:
```
l[1:]
```

[29]: [2, 'kpop', 23.4]

[31]:
```
l[1:3]
```

[31]: [2, 'kpop']

[27]:
```
l[-3:]
```

[27]: [2, 'kpop', 23.4]

[32]:
```
l[-4:-1]
```

[32]: [1, 2, 'kpop']

[34]:
```
l[0:4:2]
```

[34]: [1, 'kpop']

[36]:
```
l[4:0:-2]
```

[36]: [23.4, 2]

[41]:
```
a=["pwskills",154660,23.45,2+5] #reversing the list
a[::-1]
```

[41]:
```
[7, 23.45, 154660, 'pwskills']
```

5)lists are Mutable:They can be modified after creation which enables you to append,remove,pop,replace,insert,extend,search,sort,iterates elements as per requirement. This makes list dynamic as it grows or shrinks as per needed also makes it adoptable to changing data requirements.

[50]:
```
#ex5)append(add) at the end of the list
l = ["leh","burj","khalifa","mountains"] #syntax: list.append(element)
l.append("rivers")
l
```

[50]:
```
['leh', 'burj', 'khalifa', 'mountains', 'rivers']
```

[57]:
```
#remove
l=[1,2,3,4] #syntax:list.remove(element)
l.remove(2)
l
```

[57]:
```
[1, 3, 4]
```

[104]:
```
#pop
l=[9,8,7] #syntax:list.pop() or pop(index value may be positive or negetive)
l.pop(-1)
l
```

[104]:
```
[9, 7]
```

[73]:
```
#replace at any index if mentioned in command
l=["Ram","leela"]
l
```

[73]:
```
['Ram', 'leela']
```

[77]:
```
l[1]="sita"
str(l)
```

[77]:
```
"['Ram', 'sita']"
```

[76]:
```
str(l).replace("sita","geeta")
```

[76]:
```
"['Ram', 'geeta']"
```

[69]:
```
#insert element at specific index
l=["live","life","as","you","want"] #syntax:list.insert(position,element)
l.insert(1,"your")
l
```

[69]: ['live', 'your', 'life', 'as', 'you', 'want']

[108]:
```
#extend:modifies original list
a=['a','b']
a.extend([1,2,3])
a
```

[108]: ['a', 'b', 1, 2, 3]

[ ]:

Q3)Describe how to access,modify and delete elements in a list with examples

[112]:
```
#ANS:list elements can be accessed by index
a=["lollipop",1,2.3,500]
a[0]
```

[112]: 'lollipop'

[113]:
```
a[-2]
```

[113]: 2.3

[114]:
```
#methods to modify the list
#append:adds object to the list
a=[1,2,3]
a.append(123)
a
```

[114]: [1, 2, 3, 123]

[115]:
```
#sort:sorts list in ascending order
z=[3,7,4,2]
z.sort()
z
```

[115]: [2, 3, 4, 7]

[118]:
```
z.sort(reverse =True) #desending order
z
```

[118]: [7, 4, 3, 2]

[119]:
```
#remove:
z=[1,3,4,6]
z.remove(4) #direct value 4 is assigned in command
z
```

[119]:
```
[1, 3, 6]
```

[122]:
```
#pop:pops index value is assigned to command whose object is to be removed
#by defalut if index value is not assigned in command it pop the last value as 2
z=[0,56,2]
z.pop(1) #here 1 is index value assigned in command to remove 56
z
```

[122]:
```
[0, 2]
```

[123]:
```
#extend:modifies original list
z=[9,0,7]
z.extend([3,4,5])
z
```

[123]:
```
[9, 0, 7, 3, 4, 5]
```

[124]:
```
#insert:insert the element at particular index
z=[1,2,3,4]
z.insert(3,[5,6,7])
z
```

[124]:
```
[1, 2, 3, [5, 6, 7], 4]
```

[ ]:

Q4)Compare and contrast tuples and lists with examples

ANS:SIMILARITIES: 1)List and tuple both are built-in datastructures in python,they are offered collection of elements/objects,they are containers that stores and manipulates a multiple elements(data) of same or different datatypes(int,bool,complex etc) in a single variable.can have duplicate elements too. They can be iterable by loops.

[129]:
```
a=[1,"ee",3.5,True,3,3,3] #list
a
```

[129]:
```
[1, 'ee', 3.5, True, 3, 3, 3]
```

[138]:
```
b=(2,"ff",4.5,False,2,2,2)#tuple
b
```

[138]:
```
(2, 'ff', 4.5, False, 2, 2, 2)
```

[ ]:
```
2)both can access elements by their index
```

[131]:
```
a[0] #list
```

[131]:
```
1
```

[139]:
```
b[1]# tuple
```

[139]:
```
'ff'
```

3)Concatenation,repetition,slicing,nesting,sorting of list can be done for both types.

[1]:
```
#LIST operations:Concatenation

L1=[1,2,3]
L2=[5,6,7]
L3=L1+L2
L3
```

[1]:
```
[1, 2, 3, 5, 6, 7]
```

[2]:
```
#nesting:
L3=[[1,2,3],[9,7,5]]
L3
```

[2]:
```
[[1, 2, 3], [9, 7, 5]]
```

[2]:
```
#Repetition
L1*3
```

[2]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

[3]:
```
#slicing
L3[2:4]
```

[3]:
```
[3, 5]
```

[4]:
```
#sorting
L4=[2,6,1,5,3]
sorted(L4)
```

[4]:
```
[1, 2, 3, 5, 6]
```

[5]:
```
#TUPLE Operation:Concatenation
T1=[1,2,3]
T2=[5,6,7]
T3=T1+T2
```

```
T3
```

[5]: [1, 2, 3, 5, 6, 7]

[1]:
```
#nesting:
T4=((1,2,3),(234,345),(32,45))
T4
```

[1]: ((1, 2, 3), (234, 345), (32, 45))

[6]:
```
#Repetition
T1*3
```

[6]: [1, 2, 3, 1, 2, 3, 1, 2, 3]

[7]:
```
#slicing
T3[2:4]
```

[7]: [3, 5]

[5]:
```
#sorted
T4=(1,2,3,4)
sorted(T4)
```

[5]: [1, 2, 3, 4]

4)Following Built-in functions are common for both the types: len():return number of elements in sequence. max():return element with largest value. min():return element with smallest value.

[10]:
```
#LIST operation:
L1=[9,67,44]
len(L1)
```

[10]: 3

[11]:
```
max(L1)
```

[11]: 67

[12]:
```
min(L1)
```

[12]: 9

[13]:
```
#TUPLE operation:
T1=[9,67,44]
len(T1)
```

[13]: 3

[14]:
```
max(T1)
```

[14]: 67

[15]:
```
min(T1)
```

[15]: 9

DIFFERENCES:Lists are mutable,possible to modify as to append,delete,insert,update the elements in the list. THE elements in lists are stored in single variable separated by commas and enclosed in square bracket[].

[25]:
```
#list operation:
L1=[34,67,89]
L1.append(80)
L1
```

[25]: [34, 67, 89, 80]

[34]:
```
l2=[12,13,14]
l2.remove(13)
l2
```

[34]: [12, 14]

[35]:
```
l2
```

[35]: [12, 14]

[36]:
```
l2.insert(1,13)
l2
```

[36]: [12, 13, 14]

[37]:
```
l2
```

[37]: [12, 13, 14]

[38]:
```
l2[2]=100 #update
l2
```

[38]: [12, 13, 100]

Tuple is immutable any operation relatedmodify results in attribute error. The elements in tuple are stored in single variable and are separated by commas and enclosed in parantheses()

[40]:
```
#tuple operation
T1=(1,2,3)
T1.append(4)
T1
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[40], line 3
      1 #tuple operation
      2 T1=(1,2,3)
----> 3 T1.append(4)
      4 T1

AttributeError: 'tuple' object has no attribute 'append'
```

[42]:
```
T1.remove(1)
t1
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[42], line 1
----> 1 T1.remove(1)
      2 t1

AttributeError: 'tuple' object has no attribute 'remove'
```

[44]:
```
T1[2]=100
T1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 T1[2]=100
      2 T1

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

[ ]:

Q5)Describe key features of sets and provide examples of their use.

ANS:set is built-in datatype and is the collection unordered elements/objects in single variable ex:set={1,2,3} The key features of set are as follows explained with examples:

[9]:
```
#1)set gives unordered, unique elements, no duplicates allowed so they are␣
↪ignored at output.
set={1,2,9,4,1,1}
set
```

[9]: {1, 2, 4, 9}

[10]:
```
set[0] #2)can't access index
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 set[0]

TypeError: 'set' object is not subscriptable
```

[11]:
```
#3)They are unchangeable means once set is created,cannot change its elements␣
↪but can remove and add elements.hence,they are mutable.
set.add(100)
set
```

[11]: {1, 2, 4, 9, 100}

[12]:
```
set.remove(2)
set
```

[12]: {1, 4, 9, 100}

[13]:
```
len(set) #to detemine how many elements in set
```

[13]: 4

[14]:
```
#opeartors used in set are:union,intersection,difference,symmetric difference
S1={1,2,3}
S2={3,4,5}
S1|S2 #UNION:All the elements
```

[14]: {1, 2, 3, 4, 5}

[15]:
```
S1&S2  #INTERSECTION:commom elements
```

[15]: {3}

[16]:
```
S1-S2  #DIFFERENCE:present in first set but not in second set
```

[16]: {1, 2}

[17]:
```
S1^S2  #SYMMETRIC DIFFERENCE:not common elements
```

[17]:
```
{1, 2, 4, 5}
```

[ ]:
```

```

Q6)Discuss the use cases of tuples and sets in python programming.

ANS:tuples are ordered collection of elements used to store multiple,duplicate elements in a single variable They are heterogeneous and immutable(unchangeable) so they are used to store data which cannot be changed ever ex:account no.,adhar card no,ATM no. here are some following opeartions used in tuple:

[1]:
```
#creating a tuple
t=(1,2,2,"monjo",True,None)
t
```

[1]:
```
(1, 2, 2, 'monjo', True, None)
```

[5]:
```
type(t) #to see the type
```

[5]:
```
tuple
```

[6]:
```
#tuples are accessed by index
t=(1,2,3,5)
t
```

[6]:
```
(1, 2, 3, 5)
```

[7]:
```
t[0]
```

[7]:
```
1
```

[9]:
```
t[-1]
```

[9]:
```
5
```

[10]:
```
#if wanted to change the value in tuple error occurs
t[0] = 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 2
      1 #if wanted to change the value in tuple error occurs
----> 2 t[0] = 5

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

[12]:
```
t.count(2) #count values of element
```

[12]: 1

[13]:
```
t.index(3) #see index of any element
```

[13]: 2

[14]:
```
len(t)
```

[14]: 4

[15]:
```
t=(1,4,6,7)
sorted(t)
```

[15]: [1, 4, 6, 7]

[16]:
```
1 in t #see if element is present in tuple
```

[16]: True

[19]:
```
#nesting of tuple
t=((1,2,3),(4,5,6))
t
```

[19]: ((1, 2, 3), (4, 5, 6))

set is unordered collection of unique elements no duplicates,dont have index cannot access.They are mutable and unchangeable. sets are used in industry like to know unqiue elements in two dataframes.here,set operations like union,insection etc are used. df1=5,8,6,2 df2=6,2,8,5

[20]:
```
#creating set
s={1,1,1,2,3}
s
```

[20]: {1, 2, 3}

[21]:
```
s[0] #cannot access index
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 s[0]

TypeError: 'set' object is not subscriptable
```

[27]:
```
s[2]=100 #can't change  once created
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[27], line 1
----> 1 s[2]=100

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

[23]:
```
#but we can add and remove
s.add(100)
s
```

[23]:
```
{1, 2, 3, 100}
```

[24]:
```
s.remove(100)
s
```

[24]:
```
{1, 2, 3}
```

[25]:
```
s.clear()
s
```

[25]:
```
set()
```

[29]:
```
#different operations  also performed:
s1={1,2,3}
s2={3,4,5}
s1|s2 #union:include all elements
```

[29]:
```
{1, 2, 3, 4, 5}
```

[30]:
```
s1&s2 #intersection:commom elements
```

[30]:
```
{3}
```

[31]:
```
s1-s2
```

[31]:
```
{1, 2}
```

[32]:
```
s1^s2
```

[32]:
```
{1, 2, 4, 5}
```

[ ]:

Q7)Describe how to add,modify and delete items in a dictionary with examples.

ANS:Dictionary is unordered collection of elements. data is stored in key value pair,keys are unique and immutable cannot change once created but values assigned to keys can be change.

[55]:
```
#create a dictionary
d={"name":"ajay","email":"ajay@gmail.com"} #key:value
d
```

[55]:
```
{'name': 'ajay', 'email': 'ajay@gmail.com'}
```

[56]:
```
d['name']
```

[56]:
```
'ajay'
```

[57]:
```
d1={(1,2):"ajay"} #key can be tuple as tuple is also immutable
d1
```

[57]:
```
{(1, 2): 'ajay'}
```

[58]:
```
d
```

[58]:
```
{'name': 'ajay', 'email': 'ajay@gmail.com'}
```

[59]:
```
d["couse"]="ds" #add key value pair
d
```

[59]:
```
{'name': 'ajay', 'email': 'ajay@gmail.com', 'couse': 'ds'}
```

[60]:
```
del d["couse"]
d
```

[60]:
```
{'name': 'ajay', 'email': 'ajay@gmail.com'}
```

[61]:
```
d.keys()
```

[61]:
```
dict_keys(['name', 'email'])
```

[62]:
```
d.values()
```

[62]:
```
dict_values(['ajay', 'ajay@gmail.com'])
```

[63]:
```
d1={"course":"ml"}
d.update(d1)
d
```

[63]:
```
{'name': 'ajay', 'email': 'ajay@gmail.com', 'course': 'ml'}
```

[64]:
```
d.clear()
d
```

[64]: {}

[65]:
```
del d
d
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[65], line 2
      1 del d
----> 2 d

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

[ ]:

Q8)Discuss the importance of dictionary keys being immutable and provide examples.

ANS:Dictionary is collection of ordered elements,changeable,no duplicate allowed data is stored in key value pair but keys are immutable but values assigned to keys can be changed. Dictionary keys must be immutable because it ensures that dicitionary can efficiently look up values based on their keys. If a key was mutable,its hash value could change,making it impossible to find associated value in dictionary.

[66]:
```
#ex
emp_record={"id":"ajay","department":"enginerring"} #id and department␣
 ↪immutable and ajay enginerring mutable
emp_record
```

[66]: {'id': 'ajay', 'department': 'enginerring'}

[67]:
```
#ex
d={"name":"alice","age":30}
d
```

[67]: {'name': 'alice', 'age': 30}

[ ]: