8.DataType

April 28, 2020

Any Doubt Till Now ?

[1]:
```
jadu = [
    [
        [
            'ghor',
            [
                'andhera',
                [
                    'jadu',
                    [
                        'now',
                        [
                            'tell'
                        ],
                        [
                            'me',
                            'what'
                        ]
                    ]
                ]

            ]
        ]
      ]

    ]
```

[8]:
```
jadu[0][0][1][1][1][2][1]
```

[8]:
```
'what'
```

[10]:
```
jadu = [
        ['ghor', 'andhera',], # 0,  -6
        ['jadu'], # 1, -5
        [], # 2, -4
        ['now', 'tell', 'me'], # 3, -3
        ['what'], # 4, -2
```

```
        [] # 5, -1
      ]
```

[14]:
```
jadu[4][0]
```

[14]: 'what'

[15]:
```
jadu[-2][-1]
```

[15]: 'what'

[19]:
```
data =  [
        [
        ], # jaipur 0
        [    # total, 'died', 'recover'
            [ 1000, 400, 345], # rular
            [ 1300, 300, 250]# city
        ], # delhi 1
    ]

# print no of died patiend in delhic ity area

data[1][1][1]
```

[19]: 300

[29]:
```
'hello' >= 'hi'
```

[29]: False

[32]:
```
[ 1, 2, 3] > [ 0, 100, 200]
```

[32]: True

[30]:
```
'he' >= 'hi'
```

[30]: False

[33]:
```
l = [ 5, 4, 7, 2, 6, 3, 8]
print(l)
```

```
[5, 4, 7, 2, 6, 3, 8]
```

[36]:
```
l.sort(reverse=True)
```

[37]:
```
l
```

[37]:
```
[8, 7, 6, 5, 4, 3, 2]
```

[38]:
```
[ 4, 5] >= [ 4, 3]
```

[38]:
```
True
```

[48]:
```
# nested list or 2d list
data = [
    # name,  maths, chem, phy
    [ 'sachin',  80, 68, 79],
    [ 'sachin', 80, 68, 65],
    [ 'rajat',   65, 76, 78],
    [ 'tanvi',   95, 45, 67],
    [ 'simran',  75, 78, 67],
    [ 'kushal',  67, 45, 64],
    [ 'riya',    76, 89, 87]
    #   0,       1,  2,   3
]
```

[53]:
```
ord('A') # asscii A
```

[53]:
```
65
```

[54]:
```
ord('a') # ascii a
```

[54]:
```
97
```

[55]:
```
ord('7')
```

[55]:
```
55
```

[56]:
```
from operator import itemgetter
```

[63]:
```
data.sort(key=itemgetter(1), reverse=True)
```

[64]:
```
data
```

[64]:
```
[['tanvi', 95, 45, 67],
 ['sachin', 80, 68, 65],
 ['sachin', 80, 68, 79],
 ['riya', 76, 89, 87],
 ['simran', 75, 78, 67],
 ['kushal', 67, 45, 64],
 ['rajat', 65, 76, 78]]
```

[65]:
```
x = lambda value: value**2
```

[67]:
```
x(2)
```

[67]:
```
4
```

[82]:
```
f = lambda l:l[3]
```

[81]:
```
lang = [ 'java', 'c', 'c++', 'ruby']
f(lang)
```

[81]:
```
'ruby'
```

[79]:
```
sum([1, 2, 3, 4])
```

[79]:
```
10
```

[83]:
```
l = [ 'sachin', 40, 50, 20]
f = lambda l: sum(l[1:])
```

[84]:
```
f(l)
```

[84]:
```
110
```

[85]:
```
data
```

[85]:
```
[['tanvi', 95, 45, 67],
 ['sachin', 80, 68, 65],
 ['sachin', 80, 68, 79],
 ['riya', 76, 89, 87],
 ['simran', 75, 78, 67],
 ['kushal', 67, 45, 64],
 ['rajat', 65, 76, 78]]
```

[90]:
```
data.sort(key=lambda item: sum(item[1:]), reverse=True)
data
```

[90]:
```
[['riya', 76, 89, 87],
 ['sachin', 80, 68, 79],
 ['simran', 75, 78, 67],
 ['rajat', 65, 76, 78],
 ['sachin', 80, 68, 65],
 ['tanvi', 95, 45, 67],
 ['kushal', 67, 45, 64]]
```

[91]:
```
5[0]
```

```
␣
↪---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-91-c2ac18ab7098> in <module>
----> 1 5[0]

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

[92]:
```
1234[1]
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-92-ef4996f06adf> in <module>
----> 1 1234[1]

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

[94]:
```
 data
```

[94]:
```
[['riya', 76, 89, 87],
 ['sachin', 80, 68, 79],
 ['simran', 75, 78, 67],
 ['rajat', 65, 76, 78],
 ['sachin', 80, 68, 65],
 ['tanvi', 95, 45, 67],
 ['kushal', 67, 45, 64]]
```

[95]:
```
print(id(data))
```

```
1398064675464
```

[96]:
```
data_copy = data
```

[97]:
```
print(id(data_copy))
```

```
1398064675464
```

[98]:
```
lang = [ 'java', 'c', 'c++']
```

[99]:
```
lang_copy = lang # this is not copy in python
# here we are copying reference not data
```

[100]:
```
print(id(lang), id(lang_copy))
```

```
1398064825800 1398064825800
```

[101]:
```
lang is lang_copy
```

[101]:
```
True
```

[102]:
```
lang.append('python')
```

[103]:
```
print(lang)
```

```
['java', 'c', 'c++', 'python']
```

[104]:
```
print(lang_copy)
```

```
['java', 'c', 'c++', 'python']
```

[105]:
```
lang_copy = lang.copy() # shallow copy

# diff shallow copy & deep copy
```

[106]:
```
print(id(lang_copy), id(lang))
```

```
1398097437512 1398064825800
```

[107]:
```
lang.append('scala')
```

[108]:
```
lang
```

[108]:
```
['java', 'c', 'c++', 'python', 'scala']
```

[109]:
```
lang_copy
```

[109]:
```
['java', 'c', 'c++', 'python']
```

[110]:
```
lang
```

[110]:
```
['java', 'c', 'c++', 'python', 'scala']
```

[111]:
```
lang.reverse()
```

[112]:
```
lang
```

[112]:
```
['scala', 'python', 'c++', 'c', 'java']
```

[114]:
```
x = lang[::-1]
```

[115]:
```
x
```

[115]:
```
['java', 'c', 'c++', 'python', 'scala']
```

[116]:
```
lang.reverse() # mutable
```

[117]:
```
lang
```

[117]:
```
['java', 'c', 'c++', 'python', 'scala']
```

[118]:
```
'hello'[:3] # slicing always creates a new object
```

[118]:
```
'hel'
```

[119]:
```
[1, 5, 4, 3][:3] # slicing always creates a new object
```

[119]:
```
[1, 5, 4]
```

[120]:
```
x = [ 1, 2, 3, 4]
y = x[:]
```

[121]:
```
print(id(x), id(y))
```

```
1398095864520 1398097426248
```

[122]:
```
x.append(100)
```

[123]:
```
y
```

[123]:
```
[1, 2, 3, 4]
```

[124]:
```
x
```

[124]:
```
[1, 2, 3, 4, 100]
```

[126]:
```
print(*[ name for name in dir(list) if name[0].islower() and name[0] != '_'],␣
↪sep='\n')
```

```
append
clear
copy
count
extend
index
insert
pop
```

```
remove
reverse
sort
```

[127]:
```
lang
```

[127]:
```
['java', 'c', 'c++', 'python', 'scala']
```

[128]:
```
lang.clear()
```

[129]:
```
lang
```

[129]:
```
[]
```

[131]:
```
lang = [ 'java', 'c', 'c++']
print(id(lang))
lang = []
print(id(lang))
```

```
1398097414856
1398095931720
```

[132]:
```
lang =  [ 'java', 'c', 'c++']
print(id(lang))
lang.clear()
print(id(lang))
print(lang)
```

```
1398096432584
1398096432584
[]
```

[133]:
```
from random import randint
randint(1, 50)
```

[133]:
```
15
```

[134]:
```
x = [ randint(1, 50) for var in range(10) ]
```

[135]:
```
x
```

[135]:
```
[50, 2, 28, 41, 19, 11, 23, 15, 50, 19]
```

[136]:
```
help(x.sort)
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
```

Stable sort *IN PLACE*.

[138]:
```
x.sort(reverse=False)
x
```

[138]: [2, 11, 15, 19, 19, 23, 28, 41, 50, 50]

[142]:
```
x.sort(reverse=True)
x
```

[142]: [50, 50, 41, 28, 23, 19, 19, 15, 11, 2]

[143]:
```
x = [ 10, 50, 20, 5, 9]
```

[145]:
```
x[::-1] # step - --> Right to left
```

[145]: [9, 5, 20, 50, 10]

[146]:
```
x[::1] # step + --> left to right
```

[146]: [10, 50, 20, 5, 9]

[147]:
```
l = [ 1, 2, 3]

l1 = l

print(id(l), id(l1))
```

1398094120072 1398094120072

Dictionary

- **unordered data type** no indexing no slicing
- **map type object** it always map a value to a particuar key
- **collection of key-value pairs**
- **iterable**
- **value can be any valid python object**
- **keys are always unique in python** no duplicate key possible
- **keys should be immutable type**

syntax:

```
d = { 'key1': 'value1', 'key2': 'value2' }

{} --> denotes you are creating a dictionary same [] denotes you are creating a list

: --> seprator between key and value
```

, --> seprator between key-value pairs

[149]:
```
info = { 'name': 'sachin', 'age': 24,
       'language': [ 'hindi', 'english'],
       'country': "India"
       }
```

[150]:
```
print(type(info))
print(info)
```

```
<class 'dict'>
{'name': 'sachin', 'age': 24, 'language': ['hindi', 'english'], 'country':
'India'}
```

[153]:
```
info['name']
```

[153]: 'sachin'

[161]:
```
info['name'] = 'Sachin Yadav'
info['name']
```

[161]: 'Sachin Yadav'

[154]:
```
info['age']
```

[154]: 24

[155]:
```
info['country']
```

[155]: 'India'

[159]:
```
info['language'][0]
```

[159]: 'hindi'

[160]:
```
info['jadu'] # if key not exist raise an error
```

```
␣
↪---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call␣
↪last)

<ipython-input-160-4c24aad6089c> in <module>
----> 1 info['jadu']
```

```
KeyError: 'jadu'
```

[162]:
```
info
```

[162]:
```
{'name': 'Sachin Yadav',
 'age': 24,
 'language': ['hindi', 'english'],
 'country': 'India'}
```

[164]:
```
print(*[ name for name in dir(dict) if name[0].islower() and name[0]!= '_'],␣
 ↪sep='\n')
```

```
clear
copy
fromkeys
get
items
keys
pop
popitem
setdefault
update
values
```

[165]:
```
info
```

[165]:
```
{'name': 'Sachin Yadav',
 'age': 24,
 'language': ['hindi', 'english'],
 'country': 'India'}
```

[166]:
```
info.keys()
```

[166]:
```
dict_keys(['name', 'age', 'language', 'country'])
```

[167]:
```
info.values()
```

[167]:
```
dict_values(['Sachin Yadav', 24, ['hindi', 'english'], 'India'])
```

[168]:
```
info.items()
```

[168]:
```
dict_items([('name', 'Sachin Yadav'), ('age', 24), ('language', ['hindi',
'english']), ('country', 'India')])
```

[169]:
```
info
```

[169]:
```
{'name': 'Sachin Yadav',
 'age': 24,
 'language': ['hindi', 'english'],
 'country': 'India'}
```

[170]:
```
info_copy = info.copy()
```

[171]:
```
info_copy
```

[171]:
```
{'name': 'Sachin Yadav',
 'age': 24,
 'language': ['hindi', 'english'],
 'country': 'India'}
```

[172]:
```
info
```

[172]:
```
{'name': 'Sachin Yadav',
 'age': 24,
 'language': ['hindi', 'english'],
 'country': 'India'}
```

[173]:
```
info is info_copy
```

[173]:
```
False
```

[174]:
```
info_copy.clear()
```

[175]:
```
info_copy
```

[175]:
```
{}
```

[176]:
```
info
```

[176]:
```
{'name': 'Sachin Yadav',
 'age': 24,
 'language': ['hindi', 'english'],
 'country': 'India'}
```

[182]:
```
value = info.get('name')
print("value is ", value)
```

```
value is  Sachin Yadav
```

[187]:
```
value = info.get('alkdjf')
print("value is ", value)
```

```
value is  None
```

[185]:
```
info.get('abcd', 'no value exists')
```

[185]:
```
'no value exists'
```

[186]:
```
info.get('name', 'no value exists')
```

[186]:
```
'Sachin Yadav'
```

Data Data Type and Type Casting

[ ]:
```

```