Data Structures

Python basics

Kunal Khurana

2024-02-16

Table of contents

learning outcomes- data structures

- 1. Tuple
- 2. List
- 3. Dictionary (hash maps or associated arrays)
- 4. Set

Tuple

- cannot be changed

```
# example
tup = tuple(["foo", [1,2], True])

tup[2] = False
```

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

```
tup[1].append(3)
tup
```

```
('foo', [1, 2, 3, 3], True)
```

concatenation tuple with plus (+) operator

```
tup_2 = (4, None, 'zeal') + (5, 6, 32) + ('bar',) #no comma gives a type error
tup_2
```

```
(4, None, 'zeal', 5, 6, 32, 'bar')
```

unpacking tuples

```
seq = [(1,2,3), (4, 5, 6), (7,8,9)]

for a, b, c in seq:
    print(f'a = {a}, b = {b}, c= {c}')
```

```
a = 1, b = 2, c= 3
a = 4, b = 5, c= 6
a = 7, b = 8, c= 9
```

```
# another method
values= 1, 2, 3, 4, 5

a, b, *rest = values

rest  # used to discard
```

```
[3, 4, 5]
```

```
b
```

```
2
```

List

- same as tuples, but can be modified and lists use [ ] brackets

using 'extend' method to append already existing lists

```
x = [4, 5, 6 , None, 'foo']
x.extend([7,8, (1, 2)])

x
```

```
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
```

list concatenation with extend is faster

```
everything = []
for chunk in x:
    everything.extend(x)
    print(x)
```

```
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
```

list concatenation with (+)

```
everything = []
for chunk in x:
    everything = everything + x
    print(x)
```

```
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
[4, 5, 6, None, 'foo', 7, 8, (1, 2)]
```

slicing

```
x[-4:]
```

```
['foo', 7, 8, (1, 2)]
```

getting every element

```
x[::2] # provived elemnents till 2nd index
```

```
[4, 6, 'foo', 8]
```

```
### reversing and getting every
x [::-1]
```

```
[(1, 2), 8, 7, 'foo', None, 6, 5, 4]
```

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

Dictionaries

```
empty_dict = {}

d1 = {"a": 'some value', 'b': [1,2,3,4]}

d1
```

```
{'a': 'some value', 'b': [1, 2, 3, 4]}
```

accessing elements from the dictionary (same as list or tuple)

```
d1[3] = 'continue'
```

```
d1
```

```
{'a': 'some value', 'b': [1, 2, 3, 4], 3: 'continue'}
```

```
'b' in d1
```

```
True
```

```
del d1[3]
d1
```

```
{'a': 'some value', 'b': [1, 2, 3, 4]}
```

merge the dictionary into another using update method

```
d1.update({'d': 'food', 'e': 'à la maison'})
d1
```

```
{'a': 'some value', 'b': [1, 2, 3, 4], 'd': 'food', 'e': 'à la maison'}
```

creating dictionaries from sequences

```
mapping =  {}
for key, value in zip(x, y):
    mapping[key] = value
print(mapping)
```

```
{4: 1, 5: 2, 6: 3, None: 4}
```

```
tuples = zip(range(5), reversed(range(5)))

tuples
```

```
<zip at 0x1e1f49ed440>
```

```
mapping = dict(tuples)
mapping
```

```
{}
```

```
mapping
```

```
{}
```

```
# write a function to club the words by same first alphabet

words = ['apple', 'bat', 'bar', 'atom', 'book'] # list

by_letter = {} #empty dict

for word in words:
    letter = word[0] #first goes in
    if letter not in by_letter:
        by_letter[letter] = [word]
    else:
        by_letter[letter].append(word)

print(by_letter)
```

```
{'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

valid dictionary types

```
hash('string')
```

5928582044493709413

```
hash((1, 2 ,(2, 3)))
```

-9209053662355515447

```
hash((1, 2, [2,3])) #fails because lists are mutable
```

```
TypeError: unhashable type: 'list'
```

```
d= {}

d[tuple([1,2,3])] = 5

d
```

```
{(1, 2, 3): 5}
```

```
d[tuple('strength')] = 'persistance'

d
```

```
{(1, 2, 3): 5, ('s', 't', 'r', 'e', 'n', 'g', 't', 'h'): 'persistance'}
```

Set

- unordered collection of unique elements
- represented by curly brackets
- set operations (union, intersection, difference, and symmetric difference)
- immutable = hashable = like tuple

```
set([1, 2, 2,2,3,4,5,5,6])
```

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

```
a = {1,2,3}
b = {4,5,6}

a.union(b)
```

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

```
a | b   # means union
```

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

```
a.intersection(b)
```

```
set()
```

```
a & b # interection
```

```
set()
```

```
a.add(4)   #doesn't overwrite
a
```

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

```
a & b
```

```
{4}
```