Python Dictionaries

January 25, 2023

1 Dictionaries

- In Python dictionaries are used to maintain the data that is the form of key, value pairs
- Keys and Values a dictionary can be of any valid data type that are available in Python.
- Python dictionaries are unordered
- Examples:
  - student – branch
  - author – best seller
  - batsmen – centuries scored
  - student – list of details
  - student – marks scored in an exam
  - alphabet – it's position in latin alphabet
- In dictionaries, key-value pairs will be stored in the following format

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

- In the above example, keys are 'a', 'b', 'c' and corresponding values are 1, 2, 3 respectively.
- We can access a value associated to a key using dictionary_name[key].

```
my_dict['a']
```

- The above line is going to give 1 as output as the correspoding value of key 'a' is 1.
- If you assign a variable in Python empty curly braces {}, then automatically the data type of the variable will be a dictionary.
- Python dictionaries are not subscriptable, means they are not indexed so we cannot get the data from the dictionaries using the indexes, like we do in lists or strings.
- Dictionaries are mutable, which means we can do modifications to the data in a dictionary even after creation.

[11]:
```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict['a'])
print(my_dict['c'])
```

```
1
3
```

1.1 Ways to create a dictionary

1.1.1 Initializing on our own

[14]:
```
# initializing dictionaries
d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
cric_stat = {'sachin': 100, 'ponting': 71, 'kohli': 71}
print(cric_stat['sachin'])
print(d['d'])
print(cric_stat['kohli'])
```

```
100
4
71
```

[15]:
```
# initializing dictionaries
# dict[key] = value
d = {} # an empty dictionary
d['a'] = 1 # creating a key: value pair 'a': 1
d['b'] = 2
d['c'] = 3
d['d'] = 4
print(d)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

[17]:
```
# initializing dictionaries
# dict[key] = value
cric_stat = {}
cric_stat['sachin'] = 100
cric_stat['ponting'] = 71
cric_stat['kohli'] = 71
print(cric_stat)
print(type(cric_stat))
```

```
{'sachin': 100, 'ponting': 71, 'kohli': 71}
<class 'dict'>
```

1.1.2 Creating dictionaries interactively

- Getting the data from the user

[20]:
```
# Creating a dict that stores a student_name and total marks obtained in an exam
# How many records?
records = {}
no_of_records = int(input(f"How many records?: "))
for i in range(1, no_of_records + 1):
    name = input(f"Enter Student {i} name: ")
    marks = int(input(f"Enter Student {i} marks: "))
    records[name] = marks
```

```
print(records)
```

```
How many records?: 3
Enter Student 1 name: Rolex
Enter Student 1 marks: 400
Enter Student 2 name: Vikram
Enter Student 2 marks: 450
Enter Student 3 name: Tinal
Enter Student 3 marks: 470
{'Rolex': 400, 'Vikram': 450, 'Tinal': 470}
```

1.1.3 Writing a dictionary comprehension

[21]:
```
records = {input(): int(input()) for i in range(3)}
print(records)
```

```
Rolex
400
Vikram
450
Tina
470
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
```

[22]:
```
records = {input(f"Enter student {i + 1} name: "): int(input(f"Enter student {i␣
↪+ 1} marks: "))
           for i in range(int(input(f"How many records?: ")))}
print(records)
```

```
How many records?: 3
Enter student 1 name: Rolex
Enter student 1 marks: 400
Enter student 2 name: Vikram
Enter student 2 marks: 450
Enter student 3 name: Tina
Enter student 3 marks: 470
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
```

1.1.4 Creating a dictionary using existing data

[25]:
```
names = ['Rolex', 'Vikram', 'Tina']
# ind      0         1        2
marks = [400, 450, 470]
#ind      0    1    2
for i in range(len(names)): # 0 1 2
    print(names[i], marks[i])
```

```
Rolex 400
Vikram 450
```

```
Tina 470
```

[27]:
```
records = {}
names = ['Rolex', 'Vikram', 'Tina']
marks = [400, 450, 470]
for i in range(len(names)): # 0 1 2
    records[names[i]] = marks[i]
print(records)
```

```
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
```

[29]:
```
# using dict comprehension for the same task
names = ['Rolex', 'Vikram', 'Tina']
marks = [400, 450, 470]
records = {names[i]: marks[i] for i in range(len(names))} # i = 2
print(records)
```

```
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
```

1.1.5 Using built_in function dict()

- To dict funtion we have to send an iterable which is having key, value pairwise data in the form of tuples or lists

[30]:
```
names_and_marks = [('Rolex', 400), ('Vikram', 450), ('Tina', 470)]
records = dict(names_and_marks)
print(records)
print(type(records))
```

```
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
<class 'dict'>
```

[31]:
```
lst = [10, 20, 30]
r = dict(lst)
print(r)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [31], in <cell line: 2>()
      1 lst = [10, 20, 30]
----> 2 r = dict(lst)
      3 print(r)

TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

[1]:
```
list_of_strings = ['a1', 'b2', 'c3', 'd4', 'e5']
d = dict(list_of_strings)
print(d)
```

```
{'a': '1', 'b': '2', 'c': '3', 'd': '4', 'e': '5'}
```

[2]:
```
list_of_strings = ['a1X', 'b2Y', 'c3Z', 'd4W', 'e5V']
d = dict(list_of_strings)
print(d)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [2], in <cell line: 2>()
      1 list_of_strings = ['a1X', 'b2Y', 'c3Z', 'd4W', 'e5V']
----> 2 d = dict(list_of_strings)
      3 print(d)

ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

[32]:
```
names_and_marks = [('Rolex', 400, 'A'), ('Vikram', 450, 'O'), ('Tina', 470,␣
 ↪'S')]
records = dict(names_and_marks)
print(records)
print(type(records))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [32], in <cell line: 2>()
      1 names_and_marks = [('Rolex', 400, 'A'), ('Vikram', 450, 'O'), ('Tina',␣
 ↪470, 'S')]
----> 2 records = dict(names_and_marks)
      3 print(records)
      4 print(type(records))

ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

[33]:
```
names_and_marks = [('Rolex', (400, 'A')), ('Vikram', (450, 'O')), ('Tina',␣
 ↪(470, 'S'))]
records = dict(names_and_marks)
print(records)
print(type(records))
```

```
{'Rolex': (400, 'A'), 'Vikram': (450, 'O'), 'Tina': (470, 'S')}
<class 'dict'>
```

[34]:
```
names_and_marks = [['Rolex', [400, 'A']], ['Vikram', [450, 'O']], ['Tina',␣
 ↪[470, 'S']]]
records = dict(names_and_marks)
print(records)
print(type(records))
```

```
{'Rolex': [400, 'A'], 'Vikram': [450, 'O'], 'Tina': [470, 'S']}
<class 'dict'>
```

[3]:
```
# using dict comprehension for the same task
names = ['Rolex', 'Vikram', 'Tina']
marks = [400, 450, 470]
records = {names[i]: marks[i] for i in range(len(names))} # i = 2
print(records)
```

```
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
```

1.1.6 zip() function

- zip() takes n number of iterables, and forms a zip object that contains n length tuples (where n is the number of iterable passed). The ith value of each tuple will be taken from the ith iterable.
- When the smallest of the iterables (in terms of length) is exhausted, zip stops creating tuples.

[5]:
```
z = list(zip('abcd', '1234'))
# ('a', '1'), ('b', '2'), ('c', '3'), ('d', '4')
print(z)
```

```
[('a', '1'), ('b', '2'), ('c', '3'), ('d', '4')]
```

[7]:
```
z = dict(zip('abcd', '1234'))
# ('a', '1'), ('b', '2'), ('c', '3'), ('d', '4')
print(z)
print(type(z))
```

```
{'a': '1', 'b': '2', 'c': '3', 'd': '4'}
<class 'dict'>
```

[8]:
```
# using dict and zip for the same task
names = ['Rolex', 'Vikram', 'Tina']
marks = [400, 450, 470]
records = dict(zip(names, marks)) # ('Rolex', 400)
print(records)
```

```
{'Rolex': 400, 'Vikram': 450, 'Tina': 470}
```

[9]:
```
t = zip(range(5), 'xyz', range(4))
print(list(t))
```

```
[(0, 'x', 0), (1, 'y', 1), (2, 'z', 2)]
```

[11]:
```
t = zip(range(10, 5, -1), 'abcdefg', ['berlin', 'tokyo', 'palermo', 'lisbon',␣
↪'bogota'], range(1, 100))
print(list(t))
```

```
[(10, 'a', 'berlin', 1), (9, 'b', 'tokyo', 2), (8, 'c', 'palermo', 3), (7, 'd',
'lisbon', 4), (6, 'e', 'bogota', 5)]
```

[ ]:
```
10 9 8 7 6
a  b c d e f g
b  t p l b
1  2 3 4 5 6 7 8 9 10 11 12 .. 100
```

[ ]:
```
(10, 'a', 'berlin', 1), (9, 'b', 'tokyo', 2), (8, 'c', 'palermo', 3), (7, 'd',␣
 ↪'lisbon', 4)
(6, 'e', 'bogota', 5)
```

[16]:
```
# sub     e   m   p   ch  cs
marks = [[97, 83, 49, 26, 46],
         [38, 35, 62, 91, 69],
         [59, 37, 69, 71, 55],
         [49, 47, 56, 28, 63],
         [88, 28, 92, 76, 85]]
max_marks_student_wise = [max(i) for i in marks]
print(max_marks_student_wise)
```

```
[97, 91, 71, 63, 92]
```

[14]:
```
from random import *
marks = [[randint(25, 100) for i in range(5)] for j in range(5)]
marks
```

[14]:
```
[[97, 83, 49, 26, 46],
 [38, 35, 62, 91, 69],
 [59, 37, 69, 71, 55],
 [49, 47, 56, 28, 63],
 [88, 28, 92, 76, 85]]
```

[22]:
```
# sub     e   m   p   ch  cs
marks = [[97, 83, 49, 26, 46],
         [38, 35, 62, 91, 69],
         [59, 37, 69, 71, 55],
         [49, 47, 56, 28, 63],
         [88, 28, 92, 76, 85]]
t = list(zip(*marks))
max_marks_subject_wise = [max(i) for i in t]
print(max_marks_subject_wise)
```

```
[97, 83, 92, 91, 85]
```

[17]:
```
s = 'hello'
print(*s) # unpacking operator
```

```
h e l l o
```

[ ]:
```
'h', 'e', 'l', 'l', 'o'
```

[18]:
```
print(*marks)
```

```
[97, 83, 49, 26, 46] [38, 35, 62, 91, 69] [59, 37, 69, 71, 55] [49, 47, 56, 28,
63] [88, 28, 92, 76, 85]
```

1.2 Traversing through a dictionary

- Three different methods
  - dict.keys()
  - dict.values()
  - dict.items()

1.2.1 dict.keys()

- It returns kind of a list object that contains all the keys in dictionary.

[23]:
```
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
print(d.keys())
```

```
dict_keys(['Rolex', 'Vikram', 'Tina'])
```

[25]:
```
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
for i in d.keys():
    print(d[i])
```

```
400
450
470
```

[26]:
```
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
for i in d.keys():
    print(f'{i} --> {d[i]}')
```

```
Rolex --> 400
Vikram --> 450
Tina --> 470
```

[29]:
```
perc = {'abc': 75.50, 'xyz': 67.85, 'def': 45.70, 'ghi': 38.61}
# Print all the names whose percentage is less than 50
for i in perc.keys():
    if perc[i] < 50:
        print(i)
```

```
def
ghi
```

1.2.2 dict.values()

- return a list like object with all the values in the dictionary.

[31]:
```
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
print(d.values())
```

```
dict_values([400, 450, 470])
```

[32]:
```
# get the maximum of values in the following dict
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
print(max(d.values()))
```

```
470
```

[34]:
```
d = {'a': 1, 'b': 2, 'c': 3, 'a': 26, 'a': 7}
print(d)
```

```
{'a': 7, 'b': 2, 'c': 3}
```

[37]:
```
# dictionaries will not contain duplicate keys
cric_stat = {}
cric_stat['sachin'] = 100
cric_stat['ponting'] = 71
cric_stat['kohli'] = 71
print(cric_stat)
cric_stat['dravid'] = 60
print(cric_stat)
cric_stat['sachin'] = 101
print(cric_stat)
```

```
{'sachin': 100, 'ponting': 71, 'kohli': 71}
{'sachin': 100, 'ponting': 71, 'kohli': 71, 'dravid': 60}
{'sachin': 101, 'ponting': 71, 'kohli': 71, 'dravid': 60}
```

1.2.3 dict.items()

- Returns a list like object containing tuples of size 2.

[1]:
```
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
print(d.items())
```

```
dict_items([('Rolex', 400), ('Vikram', 450), ('Tina', 470)])
```

[2]:
```
d = {'Rolex': 400, 'Vikram': 450, 'Tina': 470}
for k, v in d.items():
    print(k, v)
```

```
Rolex 400
Vikram 450
Tina 470
```

[3]:
```
perc = {'abc': 75.50, 'xyz': 67.85, 'def': 45.70, 'ghi': 38.61}
# Print all the names whose percentage is less than 50
for k, v in perc.items():
```

```
    if v < 50:
        print(k)
```

```
def
ghi
```

1.3 Frequencies

[8]:
```
string = 'abbacddedfabcde'
d = {}
for i in string: # i = 'b'
    if i not in d.keys(): # 'b' not in d.keys()
        d[i] = 1 # d['b'] = 1
    else:
        d[i] += 1 # d['b'] += 2
print(d)
```

```
{'a': 3, 'b': 3, 'c': 2, 'd': 4, 'e': 2, 'f': 1}
```

[11]:
```
x = [3, 14, 17, 5, 2, 4, 14, 17, 15, 6, 2, 20, 4, 10, 6, 5, 9, 7, 14, 9, 13, 10,
     11, 12, 18, 1, 4, 19, 15, 5]
# Frequency of the above list
d = {}
for i in x:
    if i not in d.keys():
        d[i] = 1 # creating a new pair
    else:
        d[i] += 1 # updating an existing pair value
print(d)
```

```
{3: 1, 14: 3, 17: 2, 5: 3, 2: 2, 4: 3, 15: 2, 6: 2, 20: 1, 10: 2, 9: 2, 7: 1,
13: 1, 11: 1, 12: 1, 18: 1, 1: 1, 19: 1}
```

[14]:
```
x = [3, 14, 17, 5, 2, 4, 14, 17, 15, 6, 2, 20, 4, 10, 6, 5, 9, 7, 14, 9, 13, 10,
     11, 12, 18, 1, 4, 19, 15, 5]
# Print the numbers that appear maximum number of times in the given list
d = {}
for i in x:
    if i not in d.keys():
        d[i] = 1
    else:
        d[i] += 1
z = max(d.values())
for k, v in d.items():
    if v == z:
        print(k)
```

```
14
5
```

```
4
```

[24]:
```
n = int(input())
d = {}
for _ in range(n):
    s = input()
    if s not in d.keys():
        d[s] = 1
    else:
        d[s] += 1
print(d)
```

```
10
tokyo
rio
tokyo
professor
berlin
tokyo
berlin
professor
bogota
nairobi
{'tokyo': 3, 'rio': 1, 'professor': 2, 'berlin': 2, 'bogota': 1, 'nairobi': 1}
```

[ ]:
```
n = int(input())
d = {}
for _ in range(n):
    s = input()
    if s in d.keys():
        d[s] += 1
    else:
        d[s] = 1
print(d)
```