CSCE 110: Programming I

David Kebo Houngninou

Basics of Python

Collective Data Structures
lists, sets, dictionaries, tuples

Dictionaries

A dictionary is an unordered collection of elements.

A dictionary is a mapping between the keys and a set of values, which are mutable.

Each key maps to a value. The association of a key and a value is called a key-value pair.

Dictionaries

Keys

- are immutable.
- can be strings, numbers, and tuples.
- are unique in the dictionary.

Values

- are mutable.
- can be any data type.
- can appear repeatedly in the dictionary.

Creating dictionaries

Place elements inside curly braces {} separated by commas.

An element has a key and a value in a pair

Values can be of any data type

Keys are unique and immutable (string, number, tuple)

```
>> d = {}
>> d
   {}
>> type(d)
   <class 'dict'>
```


Creating dictionaries

Elements are separated by commas

Keys-values are separated by colons

```
-> birds = {'robin':3,'owl':[14,'apple']}
-> birds
   {'robin': 3, 'owl': [14, 'apple']}
-> birds['owl']
   [14, 'apple']
-> birds['robin']
   3
-> birds['eagle']
   Traceback (most recent call last):
     Python Shell, prompt 11, line 1
   builtins.KeyError: 'eagle'
-> birds['eagle'] = 3
-> birds
   {'robin': 3, 'owl': [14, 'apple'], 'eagle': 3}
```

Dictionary comprehension

Comprehension creates a new dictionary from an iterable.

Comprehension uses an expression pair (key: value) followed by a for statement inside curly braces {}.

```
1   even_numbers = {n: 2 * n for n in range(5)}
2   print(even_numbers)
3   even_numbers = {n: n for n in range(0, 10, 2)}
4   print(even_numbers)
5   even_numbers = {n: n for n in range(0, 10) if n % 2 == 0}
6   print(even_numbers)
```

```
> [evaluate dictionary_comprehension.py]
  {0: 0, 1: 2, 2: 4, 3: 6, 4: 8}
  {0: 0, 2: 2, 4: 4, 6: 6, 8: 8}
  {0: 0, 2: 2, 4: 4, 6: 6, 8: 8}
```


Dictionary membership

We can check if a key exists in a dictionary. Membership verification is for keys only, not values.

```
>> even_numbers = {0: "zero", 2: "two", 4: "four", 6: "six"}
>> 2 in even_numbers
   True
>> 6 not in even_numbers
   False
```

Dictionary iteration

Dictionaries support iteration using its keys.

```
>> even_numbers = {0: "zero", 2: "two", 4: "four", 6: "six"}
.. for n in even_numbers:
..     print(even_numbers[n])
..
zero
two
four
six
```


Accessing elements from a dictionary

We can access elements using keys inside square brackets [] or using the get() method.

get() returns None instead of an error if the key is not found.

```
1  specs = {"device": "computer", "memory": 32}
2  print(specs["device"])
3  print(specs["memory"])
```

```
> [evaluate access_dictionary_0.py]
computer
32
```

Accessing elements from a dictionary: get()

The method get() returns the value for the given key.

If the key does not exists, then get() returns a default value.

```
dict.get (key, default = None)
```

Parameters:

key - the key to be searched in the dictionary.

default - the value to be returned in case the key does not exist.

Accessing elements from a dictionary: get()

```
1 #Dictionary English-German - indexed by the english word
2 en_de = {"red" : "rot", "green" : "grün", "blue" : "blau", "yellow":"gelb"}
3 #Dictionary German-Portuguese - - indexed by the german word
4 de_pt = {"rot" : "vermelho", "grün" : "verde", "blau" : "azul", "gelb":"amarelo"}
5
6 print('Red in German is',en_de.get('red'))
7 print('Red in Portuguese is',de_pt.get(en_de.get('red')))
8
9 print('White in German is', en_de.get('white'))
```

```
Red in German is rot
Red in Portuguese is vermelho
White in German is None
```

Activity 4

Write a program that:

1. Creates a dictionary containing 5 scientists' last names and the year they were born.

keys: scientist's last name (e.g., Newton)

value: year a scientist was born (e.g., 1642)

2. Prints the number of items in the dictionary

Example

```
1  # initialize dictionary
2  scientists = {'Newton' : 1642, 'Turing' : 1912}
3  print('after initialization:', scientists)
4
5  # get number of items in the dictionary
6  print('number of items in dictionary:', len(scientists))
7
8  # add an item to the dictionary
9  scientists['Einstein'] = 1879
10 print('after adding Einstein:', scientists)
11
```

Changing dictionary elements

Dictionaries are mutable.

If a key already exist, the value is updated, else a new key:value pair is added.

```
1  specs = {"device": "computer", "memory": 32}
2
3  specs["memory"] = 64
4  print(specs)
5  specs["ports"] = ["USB", "HDMI"]
6  print(specs)
7  specs["ports"] = ["VGA"]
8  print(specs)
```

```
> [evaluate access_dictionary.py]
  {'device': 'computer', 'memory': 64}
  {'device': 'computer', 'memory': 64, 'ports': ['USB', 'HDMI']}
  {'device': 'computer', 'memory': 64, 'ports': ['VGA']}
```


Activity 4

3. Replace the two oldest scientists in your existing dictionary by two famous mathematicians.

Adding dictionary elements: update()

The update() method updates the dictionary with a key:value pairs, overwriting existing keys.

```
21 # check if an item is in the dictionary
22 value  = 'Curie' in scientists
23 print('Curie in dictionary?', value)
24
25 # use update to add an item to an existing dictionary
26 new_item = {'Curie' : 1867}
27 scientists.update(new_item)
28 print('after Curie update:', scientists)
```


Dictionaries: keys(), values(), items()

Dictionary methods keys(), values(), and items() return list-like values of a dictionary's keys, values, or both keys and values.

- The values returned are not true lists.
- The values cannot be modified and do not have an append() method.
- The values returned are iterable and can be used in for loops.

Dictionaries: keys(), values(), items()

The method keys() returns a list-like value of all the available keys in the dictionary.

```
12 # get the dictionary keys
13 print('keys:', scientists.keys())
14
15 # get dictionary values
16 print('values:', scientists.values())
17
18 # get all key-value pairs in the dictionary
19 print('items:', scientists.items())
20
```


Dictionaries: keys(), values(), items()

```
 1 a_dict = {'a': 14, 17:'apple'}
 2 type(a_dict.keys())
 3
 4 #print the keys
 5 for i in a_dict:
 6     print(i)
 7
 8 #Also prints the keys
 9 for i in a_dict.keys():
10     print(i)
```

```
> [evaluate dict_keys.py]
a
17
a
17
```

Dictionaries: keys(), values(), items()

The method values() returns a list-like value of all the available values in the dictionary.

```
1  a_dict = {'a': 14, 17:'apple'}
2  type(a_dict.values())
3
4  #Print the values
5  for i in a_dict.values():
6      print(i)
```

```
-> [evaluate dict_values.py]
14
apple
```


Dictionaries: keys(), values(), items()

The method items() returns a list-like of a list of dictionaries (key, value) tuple pairs.

```
1  a_dict = {'a': 14, 17:'apple'}
2
3  #Prints the keys
4  for key,val in a_dict.items():
5      print(key,val)
```

```
> [evaluate dict_items..py]
a 14
17 apple
```

Dictionaries: Sorting elements

Dictionaries are unordered sets.

The sorted() method returns a sorted list of keys in the dictionary.

```
-> birds
   {'robin': 3, 'owl': 9, 'cardinal': 10}
-> sorted(birds)
   ['cardinal', 'owl', 'robin']
-> sorted(birds.values())
   [3, 9, 10]
-> sorted(birds.items())
   [('cardinal', 10), ('owl', 9), ('robin', 3)]
```


Deleting dictionary elements

- The pop() method removes a particular element from a dictionary.
- The popitem() method removes and returns an arbitrary element (key, value) form the dictionary.
- The clear() method removes all the elements from the dictionary.
- The del keyword removes individual elements or the entire dictionary itself.

Dictionaries: delete

```
29
30 # delete an item from the dictionary
31 del scientists['Turing']
32 print('after deleting Turing:', scientists)
33
34 # remove an arbitrary item from the dictionary
35 item = scientists.popitem()
36 print('arbitrary item popped:', item)
37 print('after popitem:', scientists)
38
39 # remove all items from the dictionary
40 scientists.clear()
41 print('after clear:', scientists)
```


Dictionaries: Inverted dictionaries

Inverting a dictionary D allows you to use a value to look up a key.

An inverted dictionary $D_{inverted}$ is defined as follows:

The keys of $D_{inverted}$ are the values of D

The values of $D_{inverted}$ are the keys of D.

Exercise

Create an inverted dictionary where the keys are the number of NBA Championships and the values are the team names.

```
nba_champ  = {'Boston Celtics':17,
              'Minneapolis/Los Angeles Lakers':16,
              'Chicago Bulls':6,
              'Philadelphia/San Francisco/Golden State Warriors'
              'San Antonio Spurs':5,
              'Syracuse Nationals/Philadelphia 76ers':3,
              'Fort Wayne/Detroit Pistons':3,
              'Miami Heat':3,
              'New York Knicks':2,
              'Houston Rockets':2}
```


Solution

How many keys the inverted dictionary will have?

What are the data types of the keys and the values?

1.	Boston Celtics	17
2.	Minneapolis/Los Angeles Lakers	16
3.	Chicago Bulls	6
4.	Philadelphia/San Francisco/Golden State	5
5.	San Antonio Spurs	5
6.	Syracuse Nationals/Philadelphia 76ers	3
7.	Fort Wayne/Detroit Pistons	3
8.	Miami Heat	3
9.	New York Knicks	2
10.	Houston Rockets	2

Solution

```
nba_champ  = {'Boston Celtics':17,
              'Minneapolis/Los Angeles Lakers':16,
              'Chicago Bulls':6,
              'Philadelphia/San Francisco/Golden State Warriors':5,
              'San Antonio Spurs':5,
              'Syracuse Nationals/Philadelphia 76ers':3,
              'Fort Wayne/Detroit Pistons':3,
              'Miami Heat':3,
              'New York Knicks':2,
              'Houston Rockets':2}


# Invert the dictionary.
freq = {}
for (team, times) in nba_champ.items():
        freq[times] = freq.get(times, []) + [team]

# Print results.
for key in sorted(freq, reverse=True):
        print(key)
        for team in freq[key]:
                print(' ', team)
```


Solution

```
17
  Boston Celtics
16
  Minneapolis/Los Angeles Lakers
6
  Chicago Bulls
5
  Philadelphia/San Francisco/Golden State Warriors
  San Antonio Spurs
3
  Syracuse Nationals/Philadelphia 76ers
  Fort Wayne/Detroit Pistons
  Miami Heat
2
  New York Knicks
  Houston Rockets
```

Other dictionary methods

clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys (seq[, v])	Return a new dictionary with keys from seq and value equal to v(defaults to None).
get(key[,d])	Return the value of key. If key does not exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitrary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

Dictionaries: built-in functions

all()	Return True if all keys of the dictionary are true (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries.
sorted()	Return a new sorted list of keys in the dictionary.