Department of Computer Science and Engineering (UG Studies)

PES University, Bangalore, India

Introduction to Computing using Python (UE19CS101)

Mr. Prakash C O
Asst. Professor,
Dept. of CSE, PESU
coprakasha@pes.edu

Data Structure: dict() : dictionary :

A dictionary is a collection which is **unordered, mutable** and **indexed through keys**. In Python dictionaries are **written with curly brackets**, and they have **key-value pairs**.

```
cardict = { "brand": "Ford",
            "model": "Mustang",
            "year": 1964,
            "colors":["red", "blue", "yellow", "grey"]
          }
```

```
Knowledge = {'Frank': {'Python', 'Perl'}, 'Guido': {'Python'}, 'Monica': {'C', 'C++'}}
```

A dict is a data structure with zero or more elements with the following attributes.

- **Keys are unique. Duplicate keys are not allowed.**
- **Keys are immutable – cannot be changed**
- **Keys are hashable - Key value pairs are stored at the hashed location in some way**
- **dict like the set are unordered collection**
- **dict is indexable based on the key – key could be a string, an integer or a tuple or any immutable type**
- **An empty dict is created by using {} or by the constructor dict()**
- **We can extract keys using the method `dict.keys()` and the values using the method `dict.values()`. Both these return iterable objects. There will be one-to-one mapping between the keys in the `dict.keys()` and the values in the `dict.values()`.**
- **We can iterate through a `dict` – we actually iterate through the keys.**

Let us solve some problems and choose the right data structures.

- The input to all these problems is **a string having # of lines**.
- **Each line has a language name, a writer's name and his work.**
- We can split the string based on new line and split each of these lines based on white space and capture whatever is required.

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""
```

a) Question 1 : **find the number of books**

Solution:

Count the number of lines in the string.

```
print("# of books : ", len(all.split('\n')))
```

b) Question 2: find the number of languages

Solution:

- Split the string into lines
- Split each line and extract the first entry only
- Put them into a data structure where the entries shall be unique – that is set.
- Count them.

```
langset = set()
for line in all.split('\n'):
      #linewords= line.split()
      #langset.add(linewords[0])
      langset.add(line.split()[0])
print("# of lang : ", len(langset))
```

c) Question 3: count the number of books in each language

This is similar to the last example. But set will not be sufficient. For each language we require a count. We require a `language:count` pair where the languages are unique. The data structure for this is dict.
In these sorts of problems where a data structure has to be created, we will initialize before a loop. We build the data structure element by element within the loop. If the lang is not present, we create the lang as the key and

make the corresponding value 0 in the dict. We then count that book by adding one to the value stored in the value field for that language.

```
lang_book_count = {}    # create an empty dict
for line in all.split('\n'):
      lang = line.split()[0]
      if lang not in lang_book_count :
            lang_book_count[lang] = 0
      lang_book_count[lang] += 1

for lang in lang_book_count :
      print(lang, " => ", lang_book_count[lang])
```

 # name: 5_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find the # of books
print("# of books : ", len(all.split('\n')))
#for l in enumerate(all.split('\n')): # Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.
#     print(l)
#print()

# find the number of languages
langset = set()
for line in all.split('\n'):
      #print(line.split()[0])
      langset.add(line.split()[0])
#print(langset)
print("# of lang : ", len(langset))

# count the number of books in each language
lang_book_count = {}
for line in all.split('\n'):
      lang = line.split()[0]
      #print(lang)
      if lang not in lang_book_count :
            lang_book_count[lang] = 0
      lang_book_count[lang] += 1

for lang in lang_book_count :
```

```
	print(lang, " => ", lang_book_count[lang])
```

d) Question 4: find list of authors for each language

Names of the authors would repeat as each author may have more than one book. So the data structure shall be a dict where key is the language and the value is a set of authors.
Check the comments at the end of each line.

```
lang_author = {} # create an empty dict
for line in all.split('\n'):
	(lang, author) = line.split()[:2] # slice and pick up the first two elements
	if lang not in lang_author:       # if key lang does not exist, add that key
		lang_author[lang] = set()   # make the value an empty set
	lang_author[lang].add(author)     # add to the set uniquely
```

 # name : 6_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find list of authors for each language
# dict of sets
lang_author = {}
for line in all.split('\n'):
	(lang, author) = line.split()[:2]
#	print(lang, author)
	if lang not in lang_author:
		lang_author[lang] = set()
	lang_author[lang].add(author)

for lang in lang_author:
	print(lang)
	for author in lang_author[lang]:
		print("\t", author)
```

e) Question 5: find number of books of each author.

- The data structure shall be a dict of dict of int.
- The key for the outer dict shall be the lang.
- The key for the inner dict will be the author.
- The value shall be int. Check the comments at the end of each line.

```
lang_author = {} # empty dict
for line in all.split('\n'):
      (lang, author) = line.split()[:2]   # slice and pick up what is required
      if lang not in lang_author :        # check and create an empty dict as the value
            lang_author[lang] = {}
      if author not in lang_author[lang] : # if the key does not exist, put the key
            lang_author[lang][author] = 0  # with the value as 0
      lang_author[lang][author] += 1       # increment the count
```

 # name: 7_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find # of books of each author in each language
# soln: dict of dict of int
lang_author = {}

for line in all.split('\n'):
      (lang, author) = line.split()[:2]
      if lang not in lang_author :
            lang_author[lang] = {}
      if author not in lang_author[lang] :
            lang_author[lang][author] = 0
      lang_author[lang][author] += 1
for lang in lang_author:
      print(lang)
      for author in lang_author[lang]:
            print("\t", author, "=>",
            lang_author[lang][author])
```

f) Question 6: create a language to author to book mapping.

We will create a dict as the solution. The key will be the language. The value will be a dict. In that dict, key will be the author and the value will be a list.

Check the comments added at the end of each line.

```
Info = {} # create an empty dict
for line in all.split('\n'):
      (lang, author, title) = line.split() # get the required info by splitting
      if lang not in info :   # if lang does not exist, add that as the key in info
            info[lang] = {}
```

```
	if author not in info[lang] :     # if author does not exist, add that as the key in info[lang]
		info[lang][author] = []    # make the value an empty list
	info[lang][author].append(title) # append to the list.
```

 # name: 8_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find list of titles of each author of each lang
# soln: dict of dict of list
info = {}
for line in all.split('\n'):
	(lang, author, title) = line.split()
	if lang not in info :
		info[lang] = {}
	if author not in info[lang] :
		info[lang][author] = []
	info[lang][author].append(title)

for lang in info:
	print(lang)
	for author in info[lang]:
		print("\t", author)
		for title in info[lang][author]:
			print("\t\t", title)
```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
values()	Returns a list of all the values in the dictionary
copy()	They copy() method returns a shallow copy of the dictionary.
clear()	The clear() method removes all items from the dictionary.

`pop()`	**Removes and returns an element from a dictionary having the given key.**
`popitem()`	**Removes the arbitrary key-value pair from the dictionary and returns it as tuple.**
`get()`	**It is a conventional method to access a value for a key.**
`str()`	**Produces a printable string representation of a dictionary.**
`update()`	**Adds dictionary dict2's key-values pairs to dict**
`setdefault()`	**Set dict[key]=default if key is not already in dict**
`keys()`	**Returns list of dictionary dict's keys**
`items()`	**Returns a list of dict's (key, value) tuple pairs**
`has_key()`	**Returns true if key in dictionary dict, false otherwise**
`fromkeys()`	**Create a new dictionary with keys from seq and values set to value.**
`type()`	**Returns the type of the passed variable.**
`cmp()`	**Compares elements of both dict.**

Python Dictionaries

- A dictionary is a collection which is unordered, mutable and indexed through keys.
- In Python dictionaries are written with curly brackets, and they have key-value pairs.
- Each key-value pair in a Dictionary is separated by a colon(**:**), whereas many **`key:value`** pairs are separated by a comma.
- A Dictionary in Python works similar to the Dictionary in a real world. **Keys of a Dictionary must be unique and of *immutable* data type such as Strings, Integers and tuples**, but the values associated with keys can be repeated and be of any type.

Example

Create and print a dictionary:

```
cardict = {
            "brand": "Ford",
            "model": "Mustang",
            "year": 1964
}
print(cardict)
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
x = cardict["model"]
```

There is also a method called `get()` that will give you the same result:

Example

Get the value of the "model" key:

```
x = cardict.get("model")
```

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
cardict = { "brand": "Ford",  "model": "Mustang",  "year": 1964 }
cardict["year"] = 2018
```

Loop through a Dictionary

You can loop through a dictionary by using a `for` loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

Print all key names in the dictionary, one by one:

```
for key in cardict:
  print(key)
```

Example

Print all *values* in the dictionary, one by one:

```
for key in cardict:
  print(cardict[key])
```

Example

You can also use the values() function to return values of a dictionary:

```
for val in cardict.values():
  print(val)
```

Example

Loop through both *keys* and *values*, by using the items() function:

The items() method returns a view object. The view object contains the key-value pairs of the dictionary, as tuples in a list.

```
print(cardict.items())
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

for key, val in cardict.items():
  print(key, val)
Output:
brand Ford
model Mustang
year 1964
```

Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

Example

Check if "model" is present in the dictionary:

```
cardict = {
            "brand": "Ford",
            "model": "Mustang",
            "year": 1964
}
if "model" in cardict:
  print("Yes, 'model' is one of the keys in the cardict dictionary")
```

Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the len() method.

Example

Print the number of items in the dictionary:

```
print(len(cardict))
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
cardict["color"] = "red"
print(cardict)
```

Removing Items

There are several methods to remove items from a dictionary:

Example

The pop() method removes the item with the specified key name:

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
cardict.pop("model")
print(cardict)
```

Example

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
cardict.popitem()
print(cardict)
```

Example

The del keyword removes the item with the specified key name:

```
cardict = {
  "brand": "Ford",
```

```
  "model": "Mustang",
  "year": 1964
}
del cardict["model"]
print(cardict)
```

Example

The del keyword can also delete the dictionary completely:

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del(cardict)
print(cardict) #this will cause an error because "cardict" no longer exists.
```

Example

The clear() keyword empties the dictionary:

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
cardict.clear()
print(cardict)
```

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a *reference* to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the copy() method:

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = cardict.copy()
print(mydict)
```

Another way to make a copy is to use the built-in method dict().

Example

Make a copy of a dictionary with the dict() method:

```
cardict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(cardict)
print(mydict)
```

get() method

Definition and Usage

The `get()` method returns the value of the item with the specified key.

Syntax

```
dictionary.get(keyname, value)
```

Parameter Values

Parameter	Description
keyname	Required. The keyname of the item you want to return the value from
value	Optional. A value to return if the specified key does not exist. Default value None

More Examples

Example

Try to return the value of an item that do not exist:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.get("price", 15000)
print(x)
15000
```

pop() Method

Definition and Usage

The `pop()` method removes the specified item from the dictionary.

The value of the removed item is the return value of the `pop()` method, see example below.

Syntax

```
dictionary.pop(keyname, defaultvalue)
```

Parameter Values

Parameter	Description
keyname	Required. The keyname of the item you want to remove
defaultvalue	Optional. A value to return if the specified key do not exist. If this parameter is not specified, and the no item with the specified key is found, an error is raised

More Examples

Example

The value of the removed item is the return value of the pop() method:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.pop("model")
print(x)

#x = car.pop("color")  #KeyError: 'color'
#print(x)

x = car.pop("color","red")
print(x)
red
```

setdefault() Method

Definition and Usage

The `setdefault()` method returns the value of the item with the specified key.

If the key does not exist, insert the key, with the specified value, see example below

Syntax

```
dictionary.setdefault(keyname, value)
```

Parameter Values

Parameter	Description
keyname	Required. The keyname of the item you want to return the value from
value	Optional. If the key exist, this parameter has no effect. If the key does not exist, this value becomes the key's value Default value None

More Examples

Example

Get the value of the "color" item, if the "color" item does not exist, insert "color" with the value "white":

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("color", "white")

print(x)
```

Example

Get the value of the "model" item:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("model", "Bronco")

print(x)
```

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : { "name" : "Emil", "year" : 2004 },
  "child2" : { "name" : "Tobias", "year" : 2007 },
  "child3" : { "name" : "Linus", "year" : 2011 }
}
```

Or, if you want to nest three dictionaries that already exists as dictionaries:

Example

Create three dictionaries, than create one dictionary that will contain the other three dictionaries:

```
child_1 = { "name" : "Emil", "year" : 2004 }
child_2 = { "name" : "Tobias", "year" : 2007 }
```

```
child_3 = { "name" : "Linus", "year" : 2011 }

myfamily = {
  "child1" : child_1,
  "child2" : child_2,
  "child3" : child_3
}
```

The dict() Constructor

It is also possible to use the dict() constructor to make a new dictionary:

Example

```
cardict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(cardict)
```

Lists from Dictionaries

```
car = {  "brand": "Ford",  "model": "Mustang",   "year": 1964 }

car_view=car.items()
print(car_view)
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
print(list(car_view))
[('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)]

car_keys=car.keys()
print(car_keys)
dict_keys(['brand', 'model', 'year'])
print(list(car_keys))
['brand', 'model', 'year']

car_values=car.values()
print(car_values)
dict_values(['Ford', 'Mustang', 1964])
print(list(car_values))
['Ford', 'Mustang', 1964]
```

Turn Lists into Dictionaries

```
dishes = ["pizza", "sauerkraut", "paella", "hamburger"]
countries = ["Italy", "Germany", "Spain", "USA"]

country_specialities_iterator = zip(countries, dishes)
print(country_specialities_iterator)
<zip object at 0x7fa5f7cad408>
```

```
country_specialities = list(country_specialities_iterator)
print(country_specialities)
[('Italy', 'pizza'), ('Germany', 'sauerkraut'), ('Spain', 'paella'), ('USA','hamburger')]
```

Now our country-specific dishes are in a list form, - i.e. a list of two-tuples, where the first components are seen as keys and the second components as values - which can be automatically turned into a dictionary by casting it with dict().

```
country_specialities_dict = dict(country_specialities)
print(country_specialities_dict)
{'USA': 'hamburger', 'Germany': 'sauerkraut', 'Spain': 'paella', 'Italy': 'pizza'}
```

Yet, this is very inefficient, because we created a list of 2-tuples to turn this list into a dict. This can be done directly by applying dict to zip:

```
dishes = ["pizza", "sauerkraut", "paella", "hamburger"]
countries = ["Italy", "Germany", "Spain", "USA"]
country_specialities_dict = dict(zip(countries, dishes))
print(country_specialities_dict)
{'USA': 'hamburger', 'Germany': 'sauerkraut', 'Spain': 'paella', 'Italy': 'pizza'}
```

The zip() function take iterables (can be zero or more), makes iterator that aggregates elements based on the iterables passed, and returns an iterator of tuples.

Dict Comprehension in Python

List Comprehension is a handy and faster way to create lists in Python in just a single line of code. It helps us write easy to read for loops in a single line.

In Python, dictionary is a data structure to store data such that each element of the stored data is associated with a key. Dictionary data structure lets you query the data using key very efficiently.

The idea of comprehension is not just unique to lists in Python. Dictionaries, one of the commonly used data structures in data science, can also do comprehension. With dict comprehension or dictionary comprehension, one can easily create dictionaries.

Remember that, in python a list is defined with square brackets [] and a dictionary is defined with curly braces {}. The idea used in list comprehension is carried over in defining dict comprehension as well.

Dict comprehension is defined with a similar syntax, but with a **`key:value`** pair in expression.

```
{key:value for i in list}
```

Dict Comprehension Example 1

Let us see a quick example of creating a dict comprehension from a list of numbers.
Here let us use a list of numbers and create a dictionary with string value of the number as key and the number as values.

```
# dict comprehension to create dict with numbers as values

>>> { str(i):i  for i in [1,2,3,4,5] }

{'1': 1, '3': 3, '2': 2, '5': 5, '4': 4}
```

Dict Comprehension Example 2

Let us say, we have a list of fruits and we can use dict comprehension to create a dictionary with fruits, the list elements as the keys and the length of each string as the values.

```
# create list of fruits

>>> fruits = ['apple', 'mango', 'banana','cherry']

# dict comprehension to create dict with fruit name as keys

>>> {f:len(f) for f in fruits}

{'cherry': 6, 'mango': 5, 'apple': 5, 'banana': 6}
```

Dict Comprehension Example 3

Let us create a dictionary with dict comprehension such that elements of the list as the keys and the elements with first letter capitalized as the values.

```
>{f:f.capitalize() for f in fruits}

{'cherry': 'Cherry', 'mango': 'Mango', 'apple': 'Apple', 'banana': 'Banana'}
```

Dict Comprehension Example 4

Let us use enumerate function in dictionary comprehension. If you have not used enumerate: enumerate can take any thing iterable as input and returns element and its index.
Here we use enumerate function on the list to create index and list element tuples and use them to create a dictionary with dict comprehension. We create a dictionary with elements of the list as the keys and the index of elements as the values. Such dictionaries with element index are often useful in a variety of scenarios.

```
# dict comprehension example using enumerate function

>>>{f:i for i,f in enumerate(fruits)}

{'cherry': 3, 'mango': 1, 'apple': 0, 'banana': 2}
```

Dict Comprehension Example 5

Another use of dict comprehension is to reverse key:value in an existing dictionary. Sometimes you may want to create new dictionary from an existing directory, such that the role of key:value pair in the first dictionary is reversed in the new dictionary. We can use Dict Comprehension and flip the element to index dictionary to index to element dictionary.

```
# dict comprehension example to reverse key:value pair in a dictionary
>>>f_dict = {f:i for i,f in enumerate(fruits)}
>>>f_dict
{'apple': 0, 'banana': 2, 'cherry': 3, 'mango': 1}
# dict comprehension to reverse key:value pair in a dictionary
>>>{v:k for k,v in f_dict.items()}
{0: 'apple', 1: 'mango', 2: 'banana', 3: 'cherry'}
```

We have used dictionaries' items function to get key, value pairs in an existing dictionary and created a new dictionary where the keys in the original dictionary are values in new dictionary and vice versa.

Dict Comprehension Example 6: How To Delete Selected Keys from Dictionary using Dict Comprehension?

Let us say you have dictionary and want to create a new dictionary by removing certain key-value pair. We can use Dict Comprehension to remove selected key-value pairs from a dictionary and create a new dictionary. Let us use the "fruit" dictionary we created above.

```
fruits = ['apple', 'mango', 'banana','cherry']

f_d1 ={f:f.capitalize() for f in fruits}

f_d1
```

Let us use diction comprehension to remove two keys, apple and banana, and their values from the fruit dictionary.

```
#keys to be removed
>>>remove_this = {'apple','cherry'}
# dict comprehension example to delete key:value pairs in a dictionary
>>>{key:f_d1[key] for key in f_d1.keys() - remove_this}
{'banana': 'Banana', 'mango': 'Mango'}
```

We have removed the keys, apple and cherry, by simply using dict keys object with set operations and now the new dictionary contains just banana and mango.

References:

1. **set_dict.pdf – Prof. N S Kumar, Dept. of CSE, PES University.**
2. **https://www.w3schools.com/python/python_dictionaries.asp**
3. **https://docs.python.org/3.1/glossary.html**