Add a key: value pair to dictionary in Python

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key: value pair.

While using Dictionary, sometimes, we need to add or modify the key/value inside the dictionary. Let's see how to add a key: value pair to dictionary in Python.

Code #1: Using Subscript notation

This method will create a new key: value pair on a dictionary by assigning a value to that key.

****

```
# Python program to add a key: value pair to dictionary


dict = {'key1':'Python', 'key2':'for'}

print("Current Dict is: ", dict)


# using the subscript notation

# Dictionary_Name[New_Key_Name] = New_Key_Value


dict['key3'] = 'Python'

dict['key4'] = 'For'

dict['key5'] = 'Machine'

dict['key6'] = 'Learning'

print("Updated Dict is: ", dict)
```

****

Output:

```
Current Dict is:  {'key1': 'Python', 'key2': 'for'}

Updated Dict is:  {'key1': 'Python', 'key3': 'Python', 'key2': 'for',
'key6': 'Learning', 'key5': 'Machine', 'key4': 'For'}
```

Code #2: Using update() method

```
************************
dict = {'key1':'Pavan', 'key2':'for'}
print("Current Dict is: ", dict)

# adding dict1 (key3, key4 and key5) to dict
dict1 = {'key3':'Pavan', 'key4':'is', 'key5':'fabulous'}
dict.update(dict1)

# by assigning
dict.update(newkey1 ='portal')
print(dict)
**********************
```

Output:

```
Current Dict is:  {'key2': 'for', 'key1': 'Pavan'}
{'key2': 'for', 'newkey1': 'portal', 'key4': 'is', 'key3': 'Pavan',
'key5': 'fabulous', 'key1': 'Pavan'}
```

Code #3: Taking Key: value as input

```
**************************
# Let's add key: value to a dictionary, the functional way


# Create your dictionary class
class my_dictionary(dict):


        # __init__ function
        def __init__(self):
                self = dict()


        # Function to add key: value
        def add(self, key, value):
                self[key] = value
# Main Function
dict_obj = my_dictionary()


# Taking input key = 1, value = Pavan
dict_obj.key = input("Enter the key: ")
dict_obj.value = input("Enter the value: ")


dict_obj.add(dict_obj.key, dict_obj.value)
dict_obj.add(2, 'forPavan')
print(dict_obj)
**********************
```

Output:

```
{'1': 'Pavan', 2: 'forPavan'}
```

Python Get random dictionary pair

Sometimes, while working with dictionaries, we can have a situation in which we need to find a random pair from dictionary. This type of problem can come in games such as lotteries etc. Let's discuss certain ways in which this task can be performed.

Method #1 : Using `random.choice() + list() + items()`
The combination of above methods can be used to perform this task. The `choice` function performs the task of random value selection and `list` method is used to convert the pairs accessed using `items()` into a list over which `choice` function can work

```
# Python3 code to demonstrate working of

# Get random dictionary pair in dictionary

# Using random.choice() + list() + items()

import random

# Initialize dictionary

test_dict = {'Gfg' : 1, 'is' : 2, 'best' : 3}


# printing original dictionary

print("The original dictionary is : " + str(test_dict))


# Get random dictionary pair in dictionary

# Using random.choice() + list() + items()

res = key, val = random.choice(list(test_dict.items()))


# printing result

print("The random pair is : " + str(res))
```

Output :

```
The original dictionary is : {'Gfg': 1, 'best': 3, 'is': 2}
The random pair is : ('is', 2)
```

Method #2 : Using `popitem()`

This function is generally used to remove an item from dictionary and remove it. The logic why this function can be used to perform this task is that as ordering in dictionary is random, any pair can be popped hence random.

```
# Python3 code to demonstrate working of

# Get random dictionary pair in dictionary

# Using popitem()


# Initialize dictionary

test_dict = {'Gfg' : 1, 'is' : 2, 'best' : 3}


# printing original dictionary

print("The original dictionary is : " + str(test_dict))


# Get random dictionary pair in dictionary

# Using popitem()

res = test_dict.popitem()


# printing result

print("The random pair is : " + str(res))
```

Output :

```
The original dictionary is : {'Gfg': 1, 'best': 3, 'is': 2}

The random pair is : ('is', 2)
```

Python Convert byteString key: value pair of dictionary to String

Given a dictionary having key: value pairs as byteString, the task is to convert the key: value pair to string.

Examples:

```
Input: {b'EmplId': b'12345', b'Name': b'Paras', b'Company': b'Cyware' }
Output: {'EmplId': '12345', 'Name': 'Paras', 'Company': 'Cyware'}

Input: {b'Key1': b'Geeks', b'Key2': b'For', b'Key3': b'Geek' }
Output: {'Key1':'Geeks', 'Key2':'For', 'Key3':'Geek' }
```

Method #1: By dictionary comprehension

```
# Python Code to convert ByteString key: value

# pair of dictionary to String.


# Initialising dictionary

x = {b'EmplId': b'12345', b'Name': b'Paras', b'Company': b'Cyware'}


# Converting

x = { y.decode('ascii'): x.get(y).decode('ascii') for y in x.keys() }


# printing converted dictionary

print(x)
```

Output:

```
{'Name': 'Paras', 'EmplId': '12345', 'Company': 'Cyware'}
```

Method #2: By iterating key and values

```
# Python Code to convert ByteString key: value

# pair of dictionary to String.


# Initialising dictionary

x = {b'EmplId': b'12345', b'Name': b'Paras', b'Company': b'Cyware'}


# Initialising empty dictionary

y = {}


# Converting

for key, value in x.items():

        y[key.decode("utf-8")] = value.decode("utf-8")


# printing converted dictionary

print(y)
```

Output:

```
{'Company': 'Cyware', 'Name': 'Paras', 'EmplId': '12345'}
```

Python Convert tuple to adjacent pair dictionary

Sometimes, while working with records, we can have a problem in which we have data and need to convert to key-value dictionary using adjacent elements. This problem can have application in web development domain. Let's discuss certain ways in which this task can be performed.

Method #1 : Using `dict()` + dictionary comprehension + slicing
The above functionalities can be used to solve this problem. In this, we just slice alternate parts of tuple and assign corresponding values using dictionary comprehension. The result is converted to dictionary using `dict()`.

```
# Python3 code to demonstrate working of
# Convert tuple to adjacent pair dictionary
# using dict() + dictionary comprehension + slicing

# initialize tuple
test_tup = (1, 5, 7, 10, 13, 5)

# printing original tuple
print("The original tuple : " + str(test_tup))

# Convert tuple to adjacent pair dictionary
# using dict() + dictionary comprehension + slicing
res = dict(test_tup[idx : idx + 2] for idx in range(0, len(test_tup), 2))

# printing result
print("Dictionary converted tuple : " + str(res))
```

Output :

```
The original tuple : (1, 5, 7, 10, 13, 5)

Dictionary converted tuple : {1: 5, 13: 5, 7: 10}
```

Method #2 : Using `dict() + zip()` + slicing
This performs this task in similar way as above method. The difference is that it uses `zip()` instead of dictionary comprehension to perform task of pairing key-value pair.

```
# Python3 code to demonstrate working of
# Convert tuple to adjacent pair dictionary
# using dict() + zip() + slicing

# initialize tuple
test_tup = (1, 5, 7, 10, 13, 5)

# printing original tuple
print("The original tuple : " + str(test_tup))

# Convert tuple to adjacent pair dictionary
# using dict() + zip() + slicing
res = dict(zip(test_tup[::2], test_tup[1::2]))

# printing result
print("Dictionary converted tuple : " + str(res))
```

Output :

```
The original tuple : (1, 5, 7, 10, 13, 5)

Dictionary converted tuple : {1: 5, 13: 5, 7: 10}
```