Module 02 - 067: Python - Dictionaries, Removing items / `del dic['key']` - `.pop()` - `.clear()` - `.remove()`

Introduction

In this guide, we will explore different ways to remove items from a Python dictionary using:

- `del dict['key']`
- `.pop()`
- `.clear()`

Each method has its own use cases, advantages, and potential pitfalls.

`del dict['key']`

The `del` keyword is the simplest way to remove a key-value pair from a dictionary. However, it will raise a `KeyError` if the key does not exist.

```
teams = {
    "astros": ["Altuve", "Correa", "Bregman"],
    "angels": ["Trout", "Pujols"],
    "yankees": ["Judge", "Stanton"],
    "red sox": ["Price", "Betts"],
}

# Removing a full key-value pair
print(teams)
del teams['astros']
print(teams)  # 'astros' is removed

# Removing a specific item from a list within a dictionary
print(teams['yankees'])
del teams['yankees'][1]
print(teams['yankees'])  # 'Stanton' removed
```

Caution: If the key does not exist, `del` will raise a `KeyError`.

```
del teams['non_existent_key']  # KeyError: 'non_existent_key'
```

Best Practice: Ensure that the key exists before attempting to delete it.

`.pop()` - Safe Removal with a Default Value

The `.pop()` method allows removing a key while optionally returning a default value if the key does not exist.

```
removed_team = teams.pop('yankees', 'No team found by this name')
print(removed_team)  # Returns the removed value or default message
print(teams)  # 'yankees' key is removed
```

Advantages:

- Unlike `del`, `.pop()` does not throw an error if the key is missing.
- It returns the removed value, making it useful for debugging and logic handling.

```
removed_team = teams.pop('mets', 'No team found')
print(removed_team)  # 'No team found' since 'mets' doesn't exist
```

`.clear()` - Removing All Items from a Dictionary

If you need to empty a dictionary completely, use `.clear()`.

```
teams.clear()
print(teams)  # {}
```

Use Case: When you want to reset a dictionary without reinitializing it.

Summary of Removal Methods

Method	Removes Key	Removes Value	Error on Missing Key?	Returns Value?
`del`			(raises KeyError)	
`.pop()`			(default return)	
`.clear()`		(all)		

Best Practices:

- Use `del` only when you are sure the key exists.
- Prefer `.pop()` when you want to handle missing keys gracefully.
- Use `.clear()` when you need to empty an entire dictionary.

Code Example

```
# Example showcasing different dictionary removal methods
teams = {
    "astros": ["Altuve", "Correa", "Bregman"],
    "angels": ["Trout", "Pujols"],
    "yankees": ["Judge", "Stanton"],
    "red sox": ["Price", "Betts"],
}

# Using del
print("Original Dictionary:", teams)
del teams['astros']
print("After del:", teams)

# Using pop
removed_team = teams.pop('yankees', 'No team found')
print("Removed team:", removed_team)
print("After pop:", teams)

# Using clear
teams.clear()
print("After clear:", teams)
```

Python Documentation Reference: Dictionaries

Video lesson Speech

So far in this section on Python dictionaries, we've walked through a number of ways that we can work with this type of a key-value based data structure and in this guide we're going to see how we can remove elements from a dictionary.

There are going to be many different ways to do it, I'm going to give you the pros and cons of using them.

large

Figure 1: large

large

Figure 2: large

`del dict['key']`

The first one is the most basic way and that is simply by deleting the item with the `del` short for delete keyword and the syntax for this is you're going to pass the name of the dictionary and then use the dictionary lookup syntax here.

So, if I say that I want to delete the Astro's then I can run this and you can see here in the output that we no longer have the Astro's listed here in the list of key-value pairs.

Now, this works perfectly fine and for many circumstances such as when you know with 100% certainty that that key exists in the dictionary.

This is going to work.

Now I'm going to show you when this is going to cause an error and that is when you try to perform delete on a key that doesn't exist so if I type the Mets in here which does not exist in the list of keys you're going to see that I get a key error.

Now, this is very similar to looking up an element, and if you remember back to when we talked about the `get` function and talked about the differences and we can do it right here just for the sake of review.

So, if I try to print out teams by performing a lookup on Mets I'm going to get a key error

and if you remember **the fix for that is by using the get function**.

I can say get and then provide a default value so I can say "no team found by that name" and that is going to give me a fixed type of output so it's not going to throw an error it's simply going to give me that value of no team found by that name.

So, just like we have this get function.

.pop()

We also have a similar process for when we want to remove items and it's called pop and so I'm going to just get rid of the entire line

```
print(teams.get('mets', 'No team found by that name'))
```

And let's see how we can use `pop`.

With Pop, you simply call teams.pop, and then you pass in in a function calls. We're going to use parens here you pass in the name of the element that you want to remove from the dictionary so I can pass in that name and then pass in a default value if that name isn't found. We'll say "no team found by that name" and if I run this now you're going to see that it performed that deletion so we no longer have Astro's here in the team dictionary.

Now, what happens if I type in a team and a key that doesn't exist? If I run this now you'll see that everything works and we get no output.

That may be a little bit of curiosity to you if you want to know where exactly is this 'No team found by that name' going?

The answer is that **Pop has a very cool little secret** with it and that is that it returns a value.

It returns either the value of the Look-Up or it returns the value of this default element here(our default element is 'No team found by that name').

So, if you come here, in order to access this you need to either print it out or store it in a variable.

So I'm going to say removed_team and set equal to teams.pop everything else stays the same.

large

Figure 3: large

large

Figure 4: large

large

Figure 5: large

And now let's just at the very bottom print out remove_team. Now if I run us you'll see that no team found by that name is printed out.

So, that is the way that pop works.

And if I come over here and I say Yankees and run it again. Here you go. Right now the value.

Now, **keep in mind this is not the key and value**.

Notice it doesn't say Yankees it's simply the value of the Look-Up.

So pop works in a very similar way as the get function with the key difference simply being that it also has the side effect of removing that element from the dictionary and so whenever I'm working with a dictionary in python and I want to remove an element and I want to have access to it and have the ability to have a default in case there was not a key by that name inside the dictionary then pop is a great option so I highly recommend for you to experiment with using both delete and pop to see which one you prefer.

.clear()

And you're going to discover that in your own programs you're going to utilize both of them like I mentioned `del` gives you the nice benefit it's a very fast and as long as you know with 100% certainty that that key exists. It's perfectly fine to use whereas `pop` gives you a number of extra added features.

Code

```python
# 02-067: Removing Key/value items

teams = {
    "astros" : ["Altuve", "Correa", "Bregman"],
    "angels":  ["Trout", "Pujols"],
    "yankees": ["Judge", "Stanton"],
    "red sox": ["Price", "Betts"],
}
print('\nOriginal list : \n'f'{teams}')


# 'del dict['key'][value Index]
print('\nUsing del:\n')
del teams['astros']          # astros entire key
print(teams)

del teams['yankees'][1]      # yankees key, Stanton value
print(teams)

#del teams['asdasdasd']       # A non-existent key will throw an error
```

large

Figure 6: large

large

Figure 7: large

large

Figure 8: large

```
#print(teams)


# .get('if', 'else) avoids error at non-existent values
print('\nUsing .get()\n')
print(teams.get('angels', 'No key found for this value'))
print(teams.get('asdasdasd', 'No key found for this value'))



# .pop('key', 'new value for the key)
print('\nUsing .pop()\n')

extracted = teams.pop('astros', 'No key found')

print(teams)
print(extracted)    # No key found due to .pop() removals

# .pop() with non-existant key

extracted_2 = teams.pop('sasdasd', 'Non.existant value, and the dictionary has NOT been altered')
print(teams)
print(extracted_2)


removed_team = teams.pop('rays', 'No team found by this name')
print(teams)
print(removed_team)


removed_team_2 = teams.pop('yankees', 'No team found by this name')
print(teams)
print(removed_team_2)   # The key exists, so .pop() do its stuff


# .clear() !

teams = {
    "astros" : ["Altuve", "Correa", "Bregman"],
    "angels":  ["Trout", "Pujols"],
    "yankees": ["Judge", "Stanton"],
    "red sox": ["Price", "Betts"],
}

print('\n\nOriginal Dictionary :\n'f'{teams}')
print(teams.clear())
print(teams)            # {}
```

IMG

Figure 9: IMG