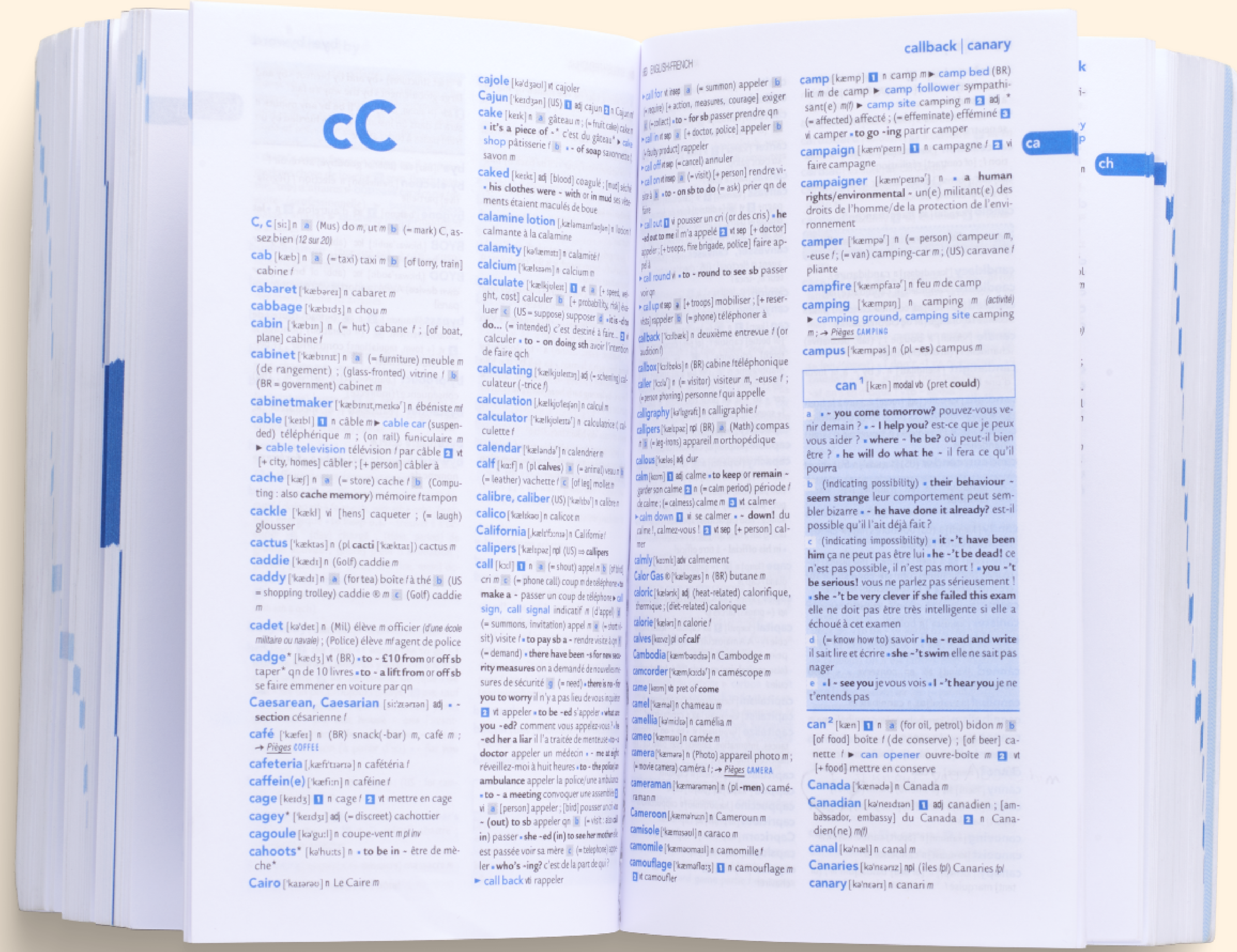
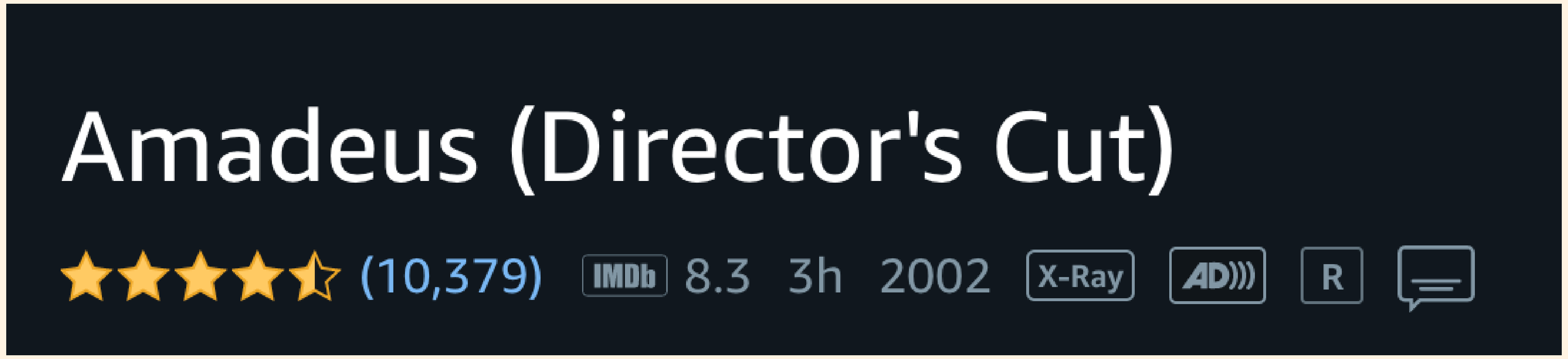


Dictionaries

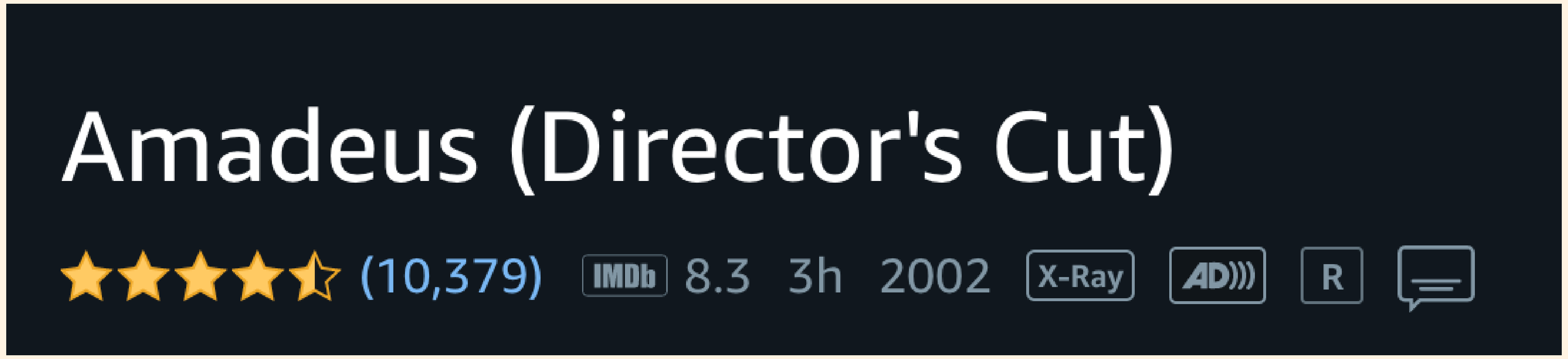
How Would We Store This?



Amadeus (Director's Cut)

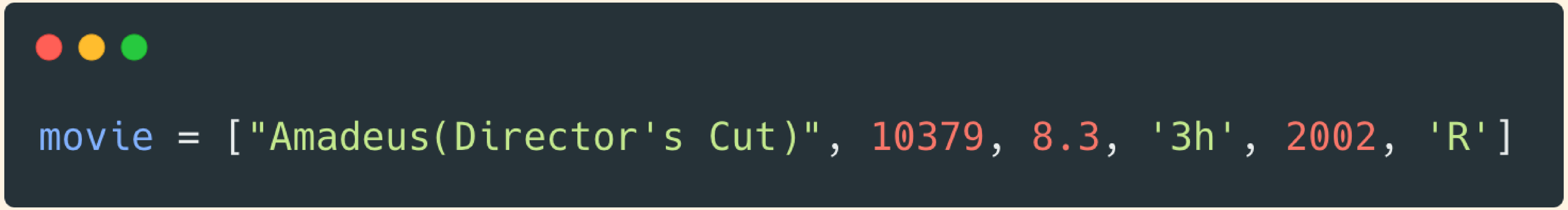
★★★★⯪ (10,379) IMDb 8.3 3h 2002 X-Ray AD))) R

How Would We Store This?



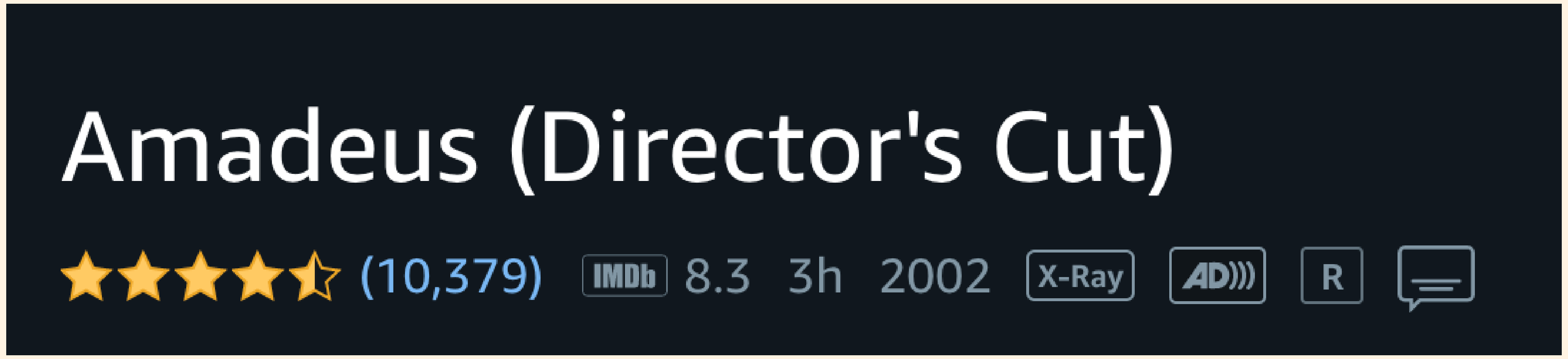
Amazeus (Director's Cut)

★★★★⯪ (10,379) IMDb 8.3 3h 2002 X-Ray AD))) R



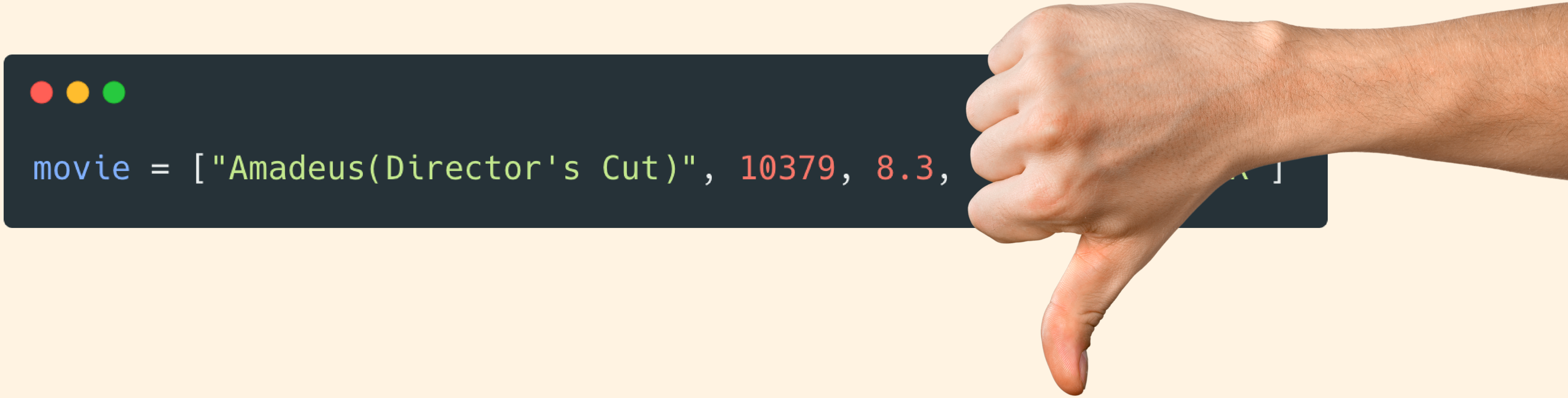
```python
movie = ["Amadeus(Director's Cut)", 10379, 8.3, '3h', 2002, 'R']
```

How Would We Store This?

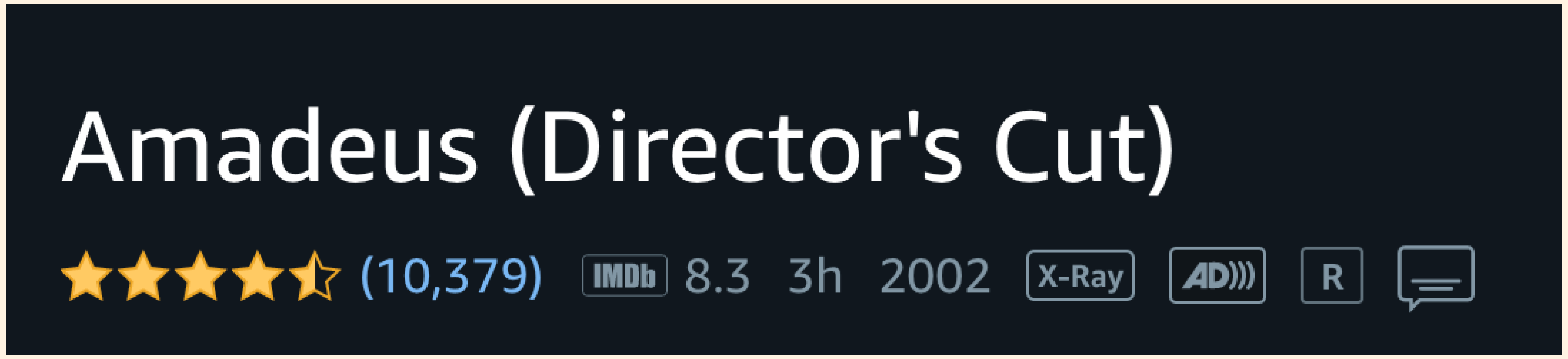


Amadeus (Director's Cut)

★★★★⯪ (10,379) IMDb 8.3 3h 2002 X-Ray AD))) R

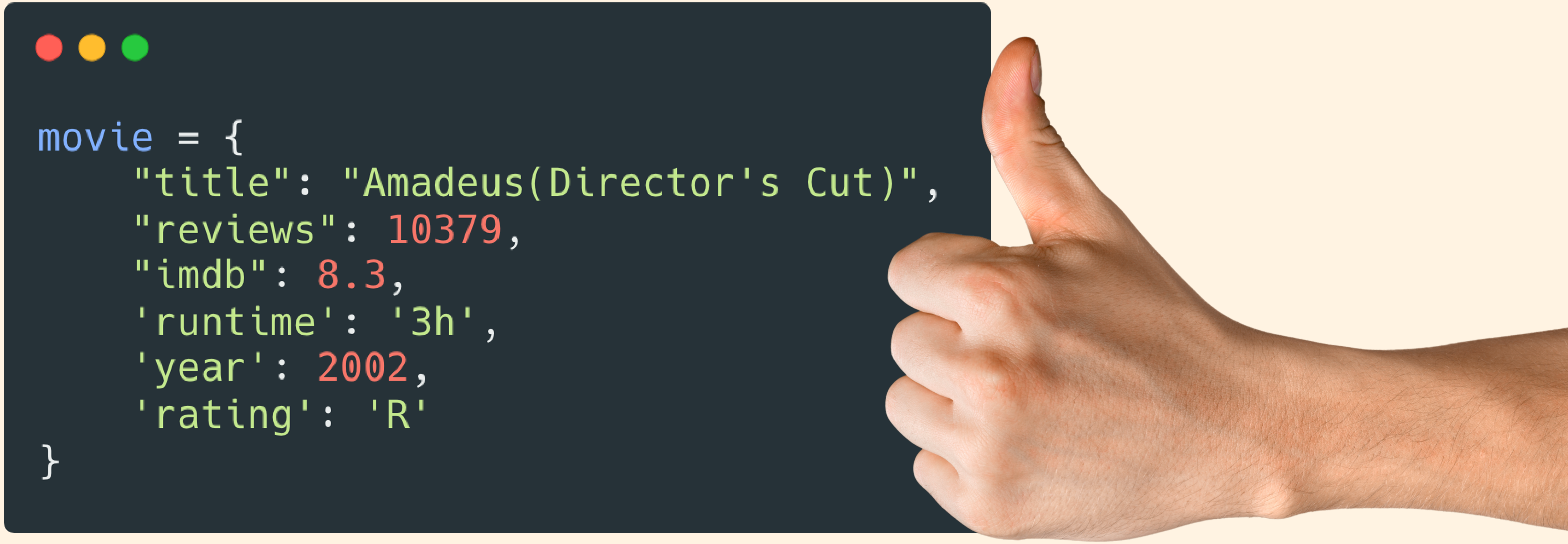


```
movie = ["Amadeus(Director's Cut)", 10379, 8.3, [illegible] ]
```



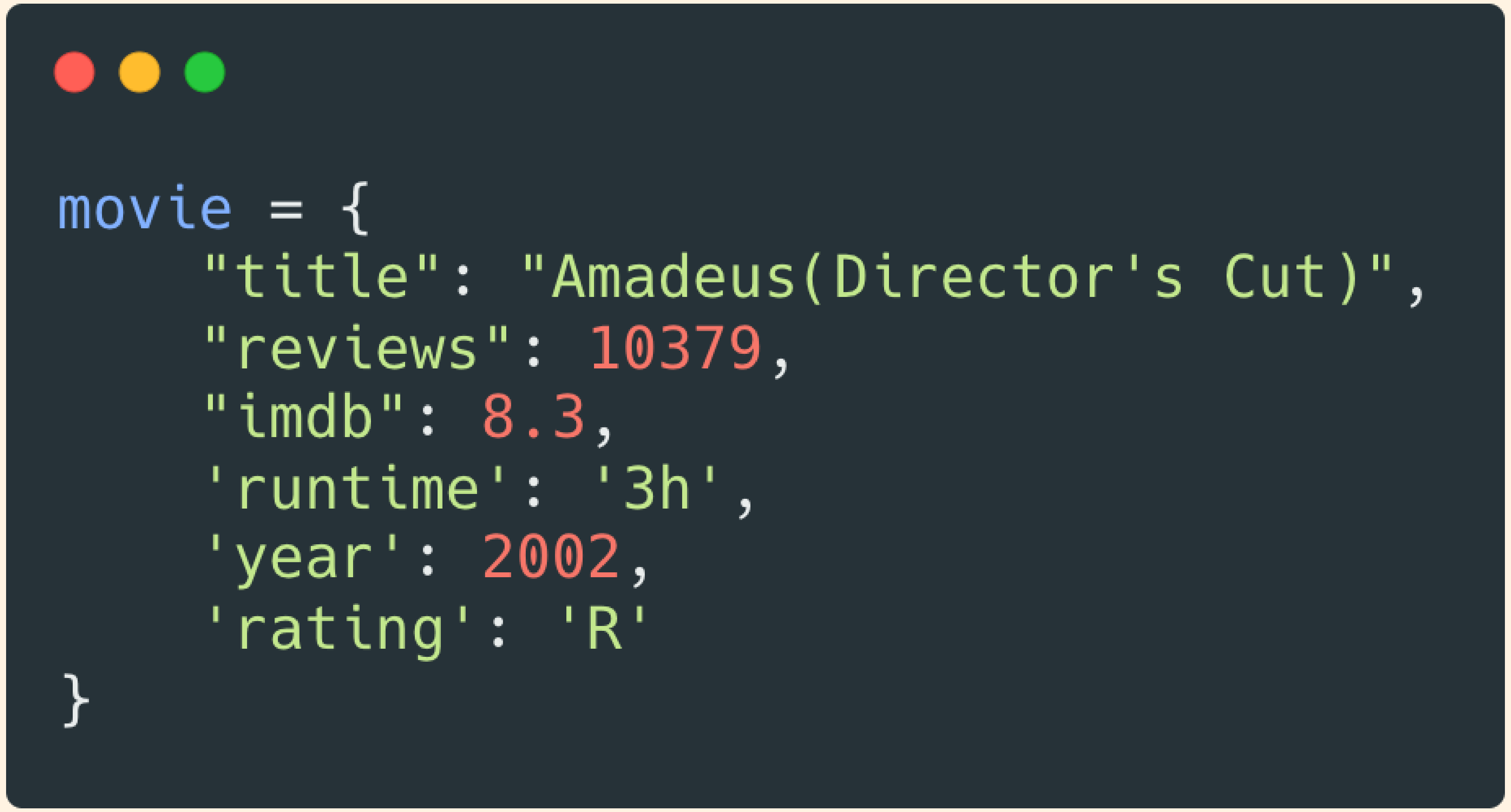
Amadeus (Director's Cut)

★★★★☆ (10,379) IMDb 8.3 3h 2002 X-Ray AD))) R

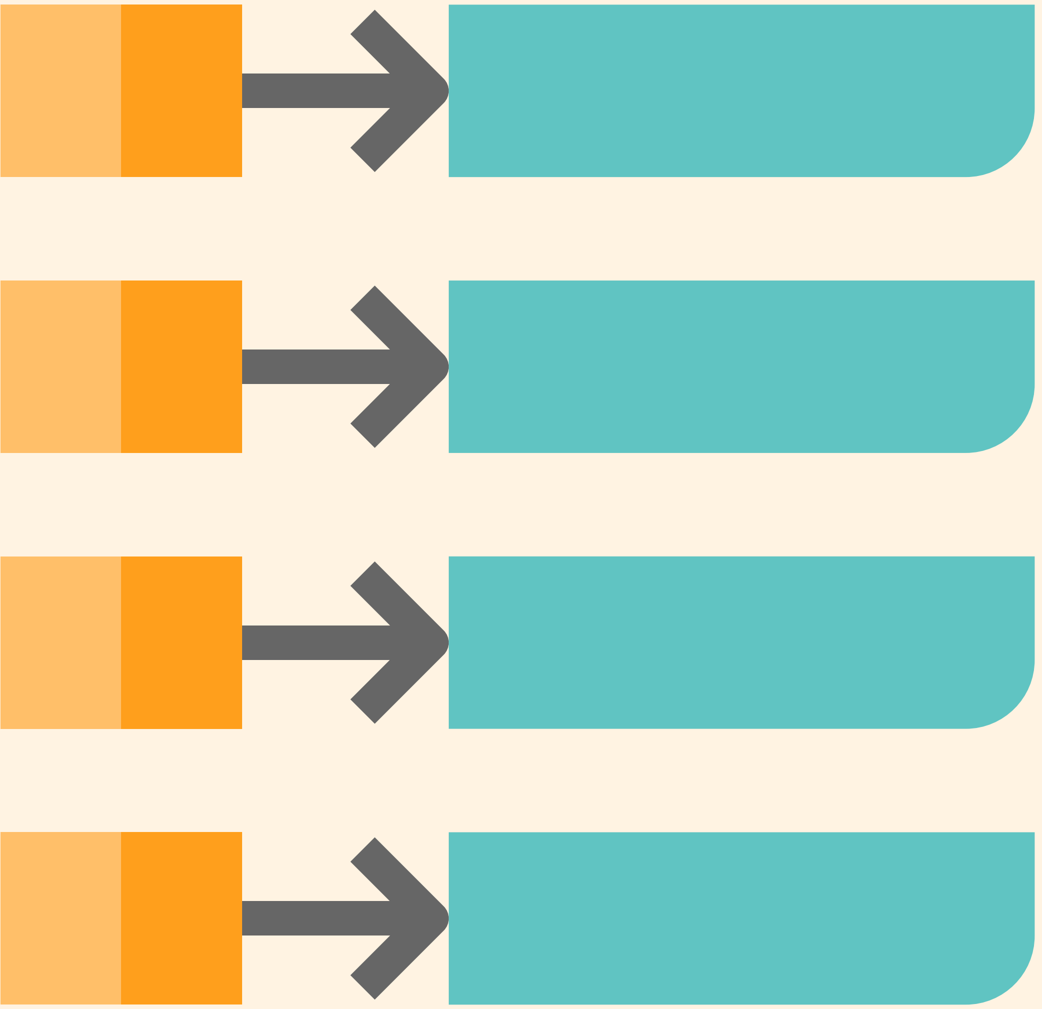


```
movie = {
    "title": "Amadeus(Director's Cut)",
    "reviews": 10379,
    "imdb": 8.3,
    'runtime': '3h',
    'year': 2002,
    'rating': 'R'
}
```

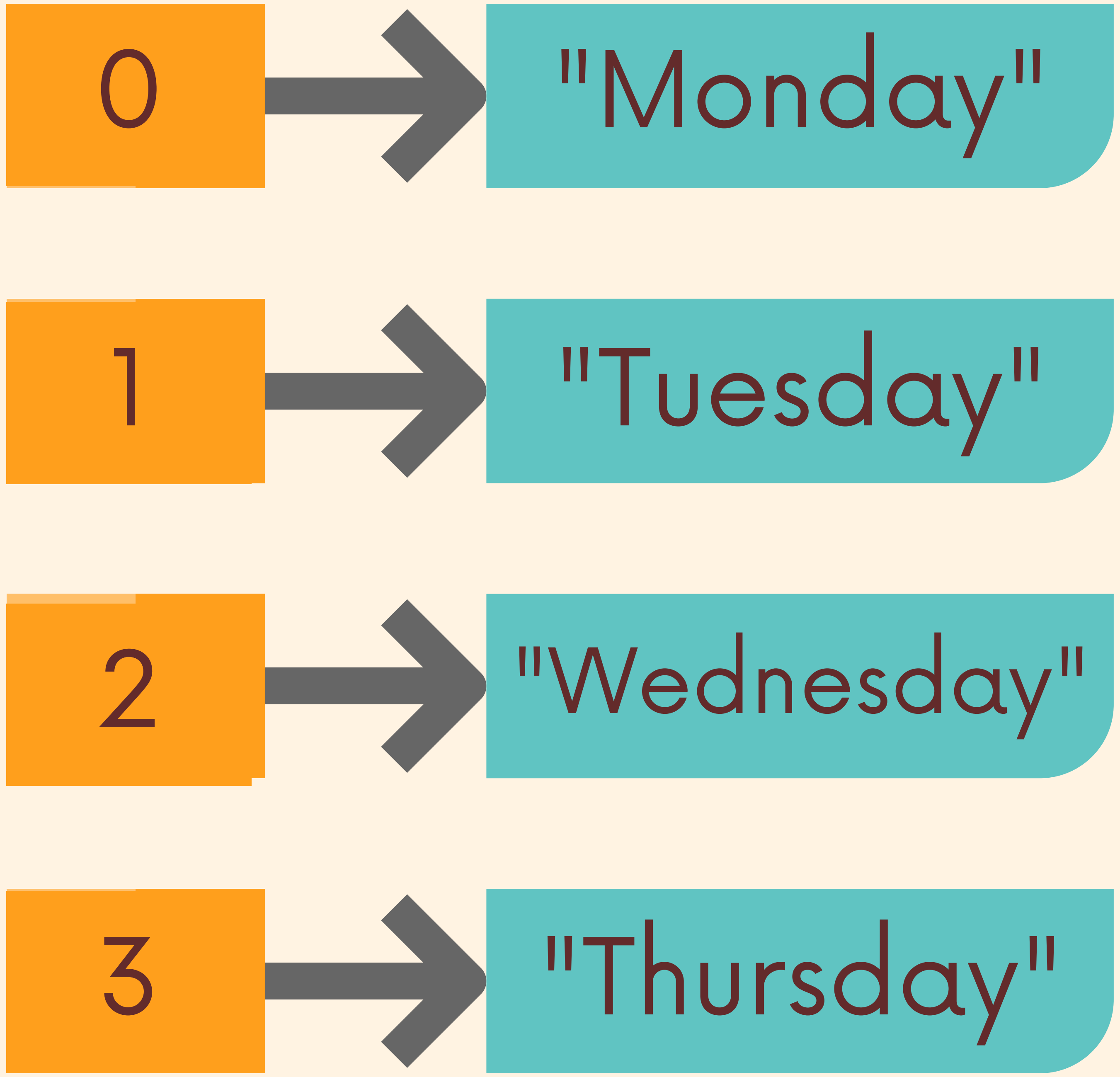
Key-Value Pairs



```
movie = {
    "title": "Amadeus(Director's Cut)",
    "reviews": 10379,
    "imdb": 8.3,
    'runtime': '3h',
    'year': 2002,
    'rating': 'R'
}
```



Index-Value Pairs



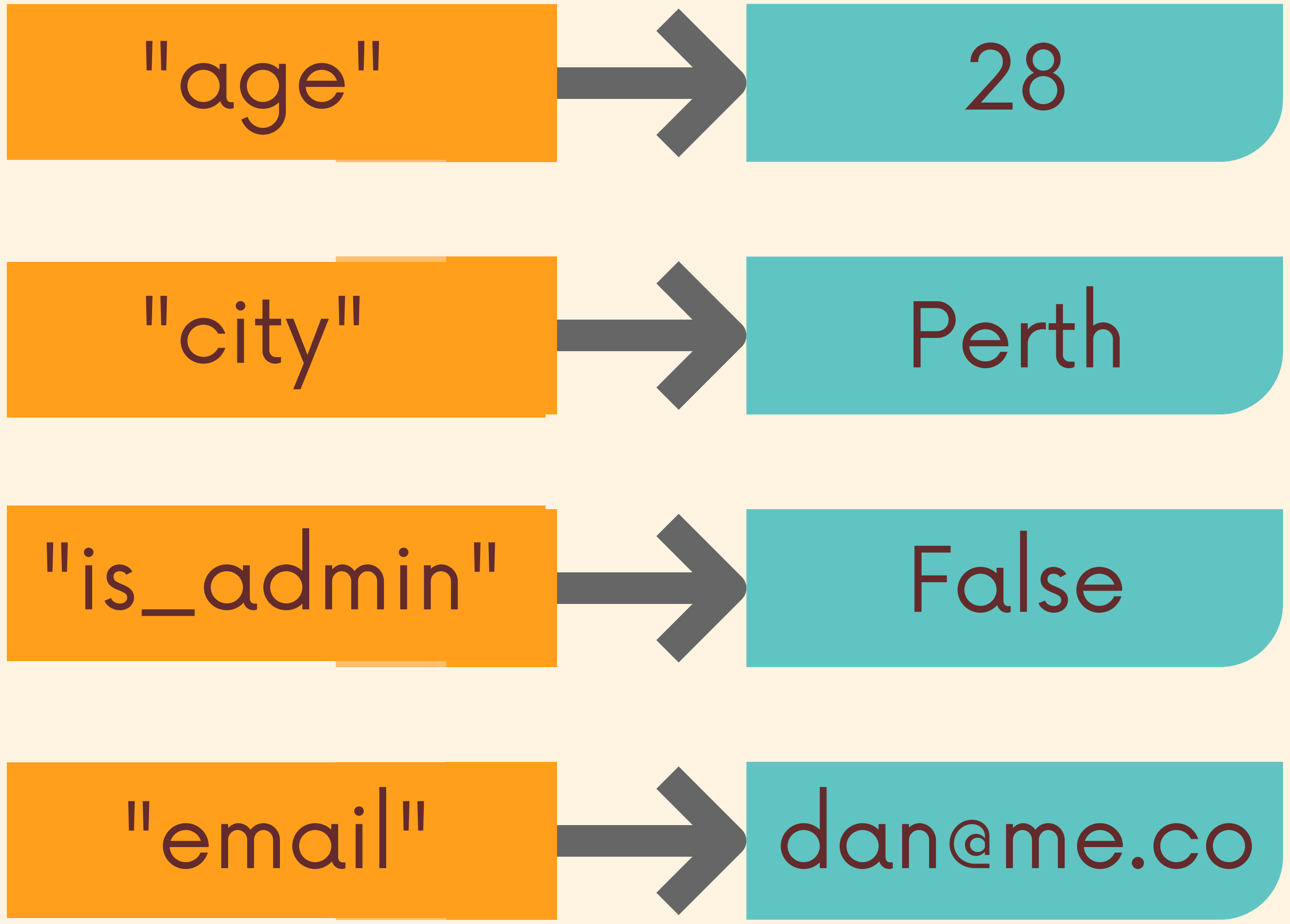
0 → "Monday"

1 → "Tuesday"

2 → "Wednesday"

3 → "Thursday"

Key-Value Pairs

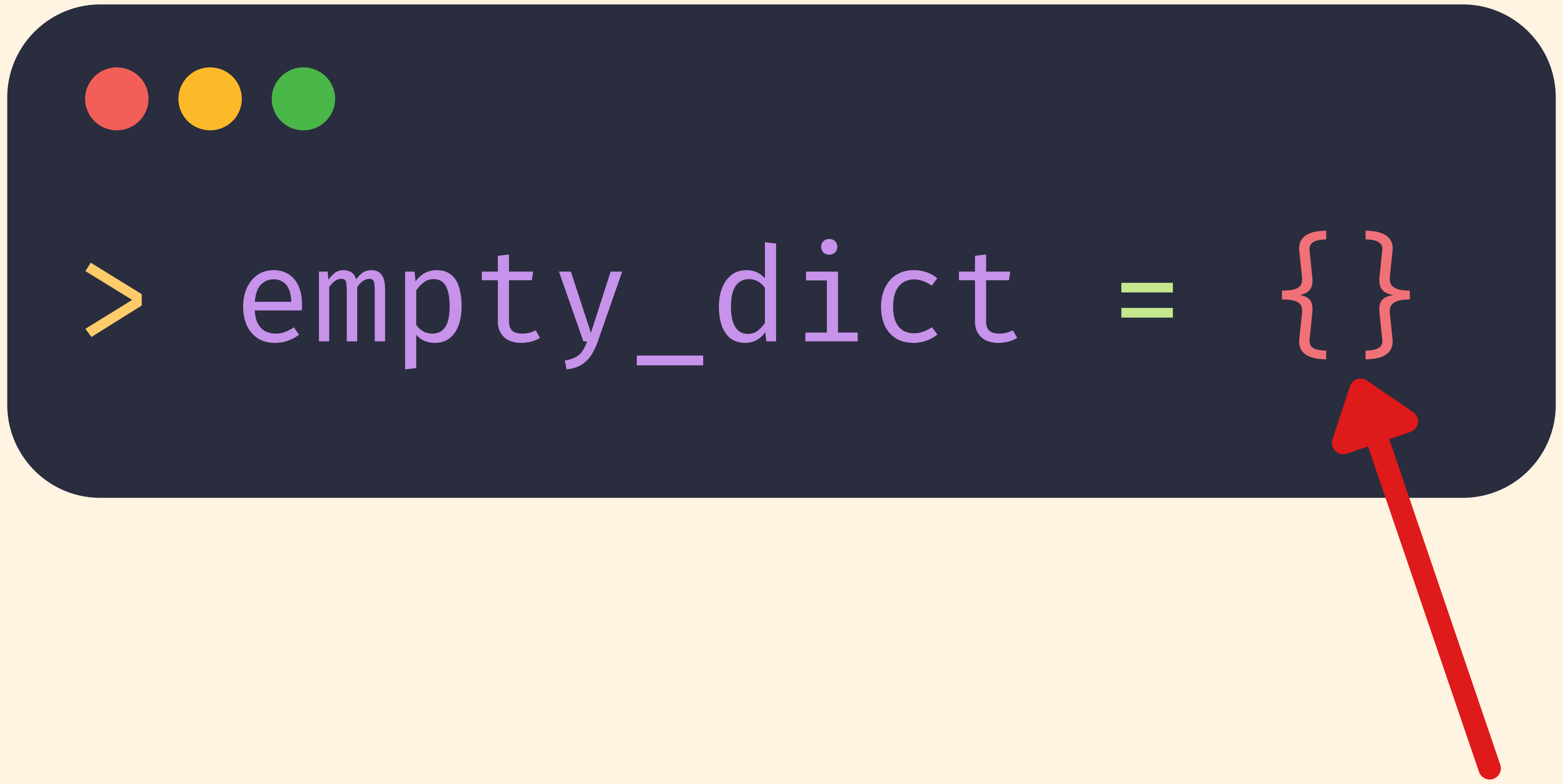


"age"	28
"city"	Perth
"is_admin"	False
"email"	dan@me.co

Dictionaries

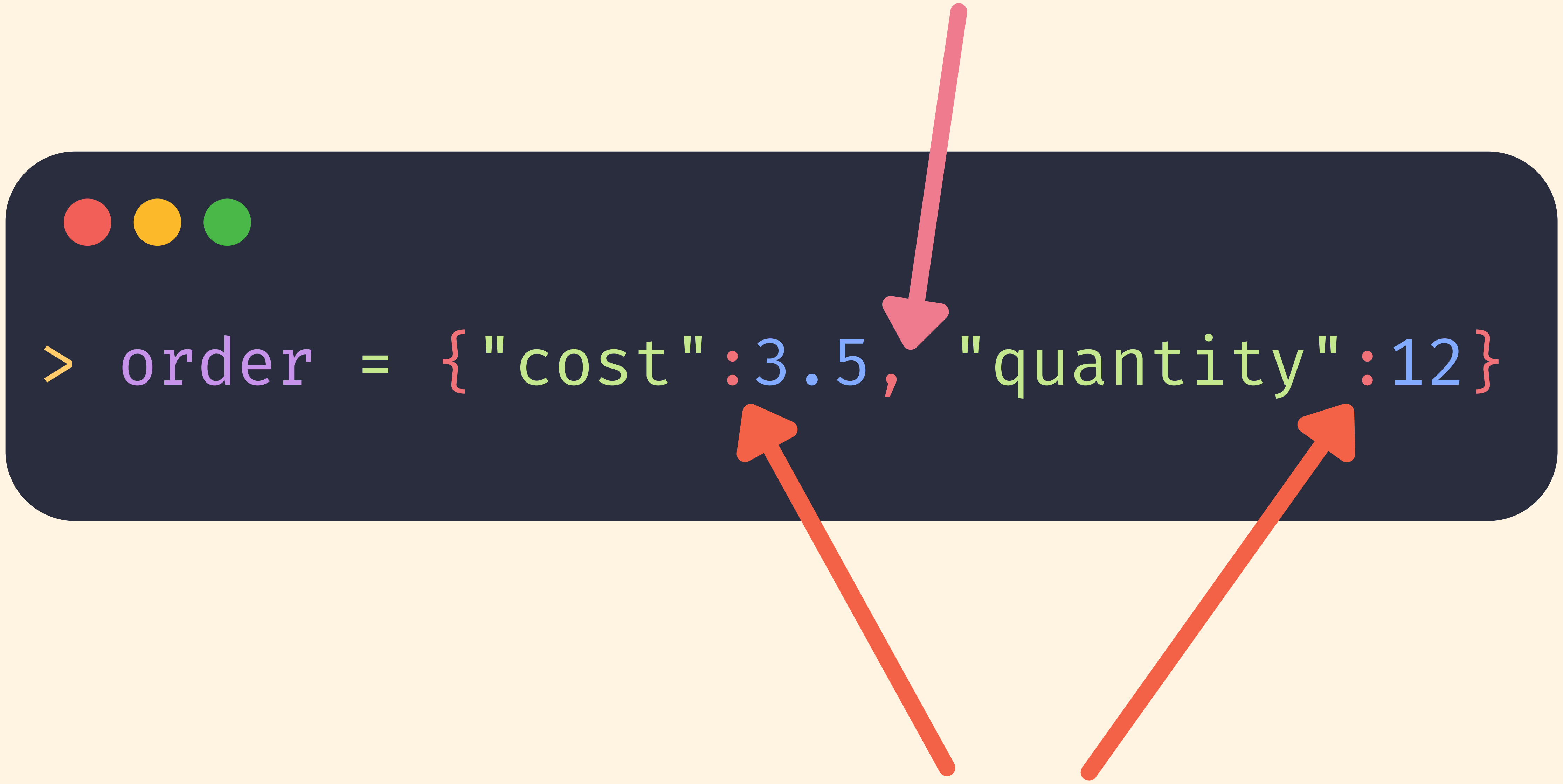
Dictionaries, known as associative arrays in some other languages, are **indexed by keys** rather than a numerical index

- A dictionary holds key-value pairs
- Keys can be any immutable type: numbers, strings, booleans, etc.
- Values can be whatever you want!



```
> empty_dict = {}
```

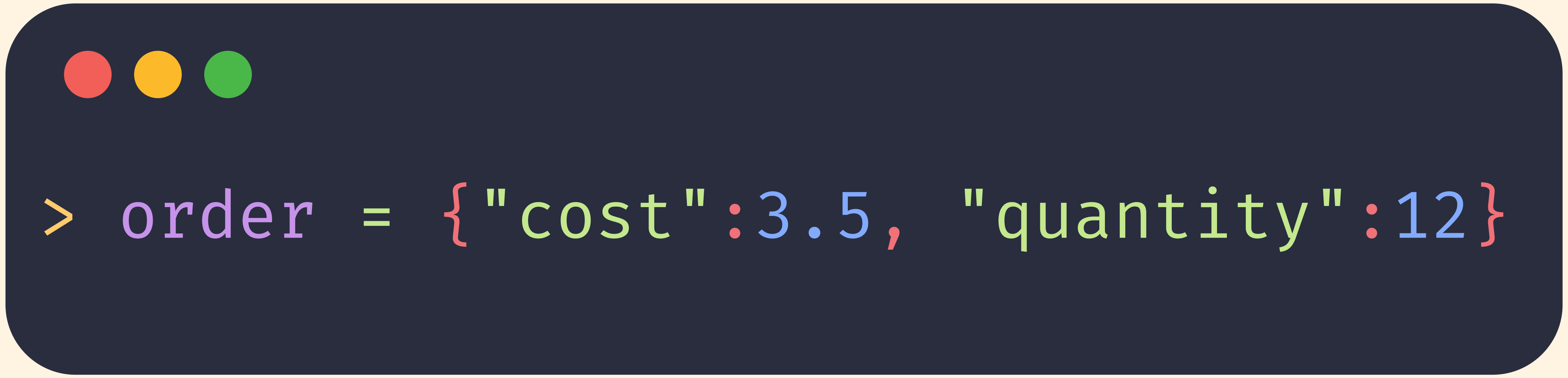
Curly Braces



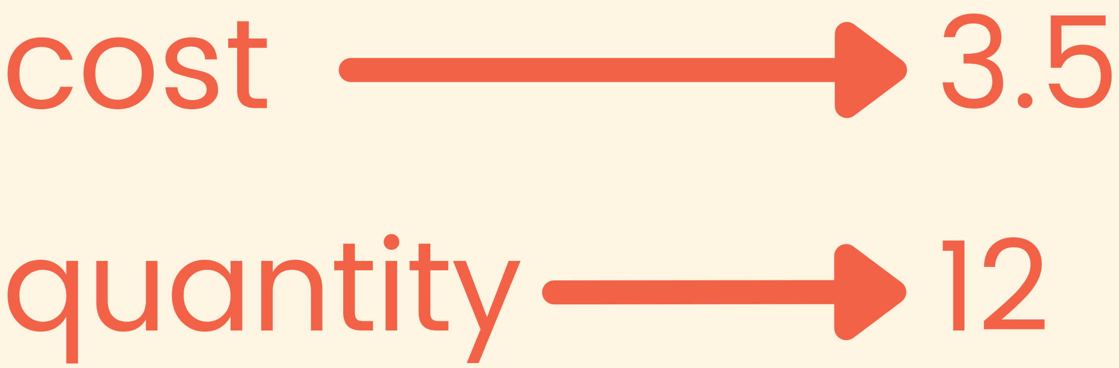
```
> order = {"cost":3.5, "quantity":12}
```

Comma

Colons



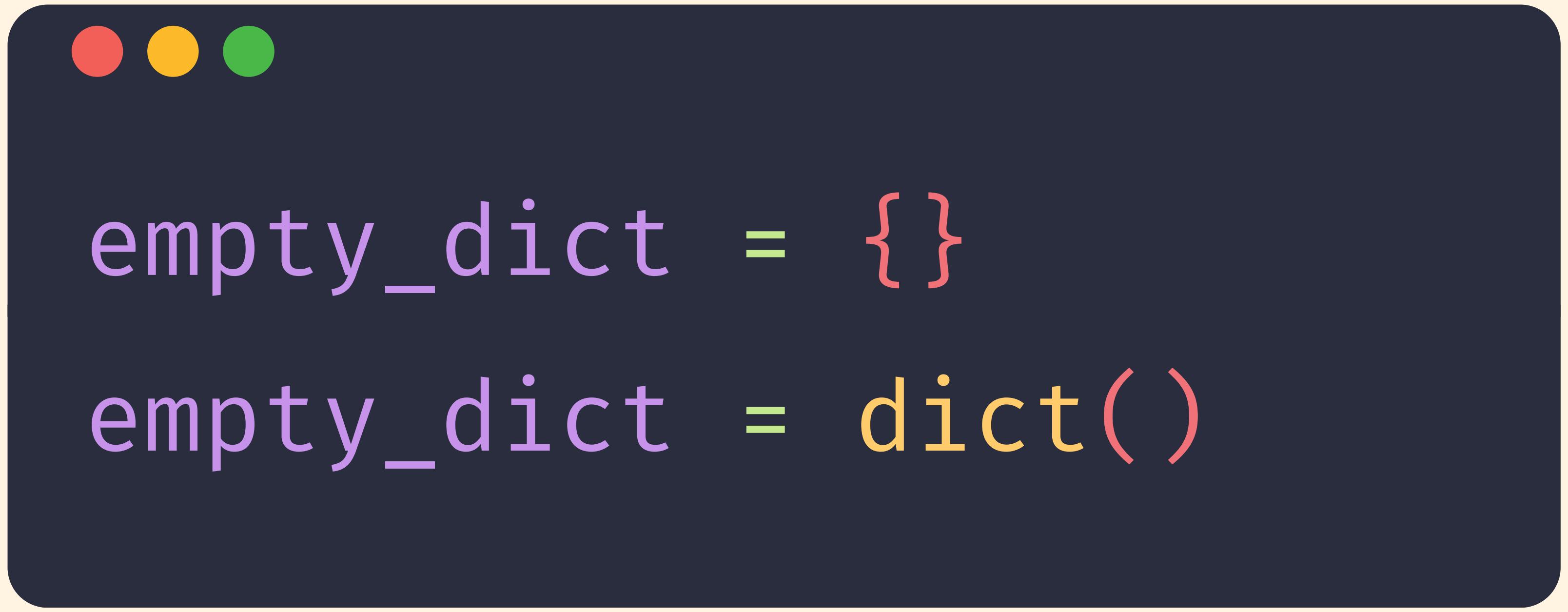
```
> order = {"cost":3.5, "quantity":12}
```



cost → 3.5

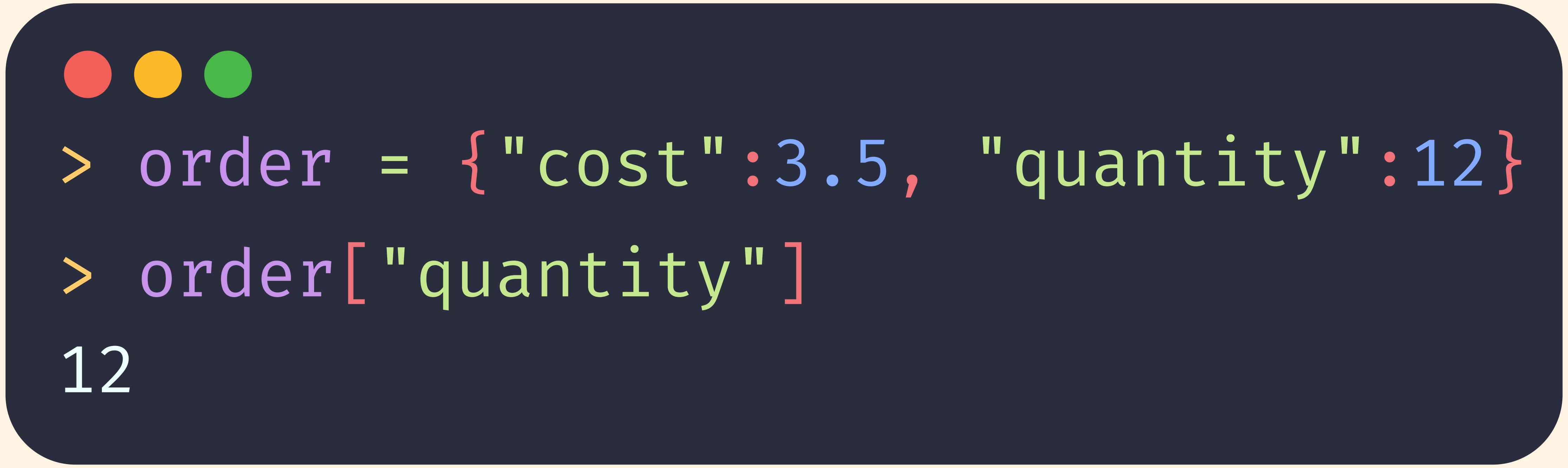
quantity → 12

Empty Dicts

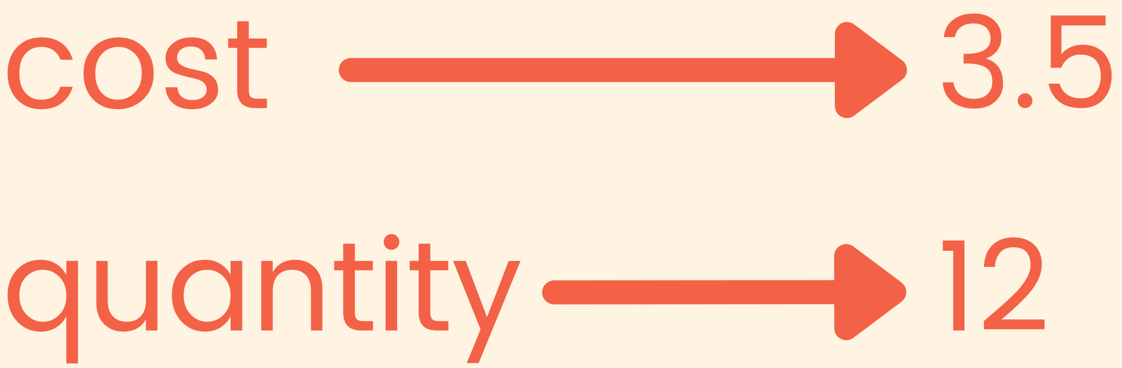


```
empty_dict = {}
empty_dict = dict()
```

retrieve values using **dict[key]**



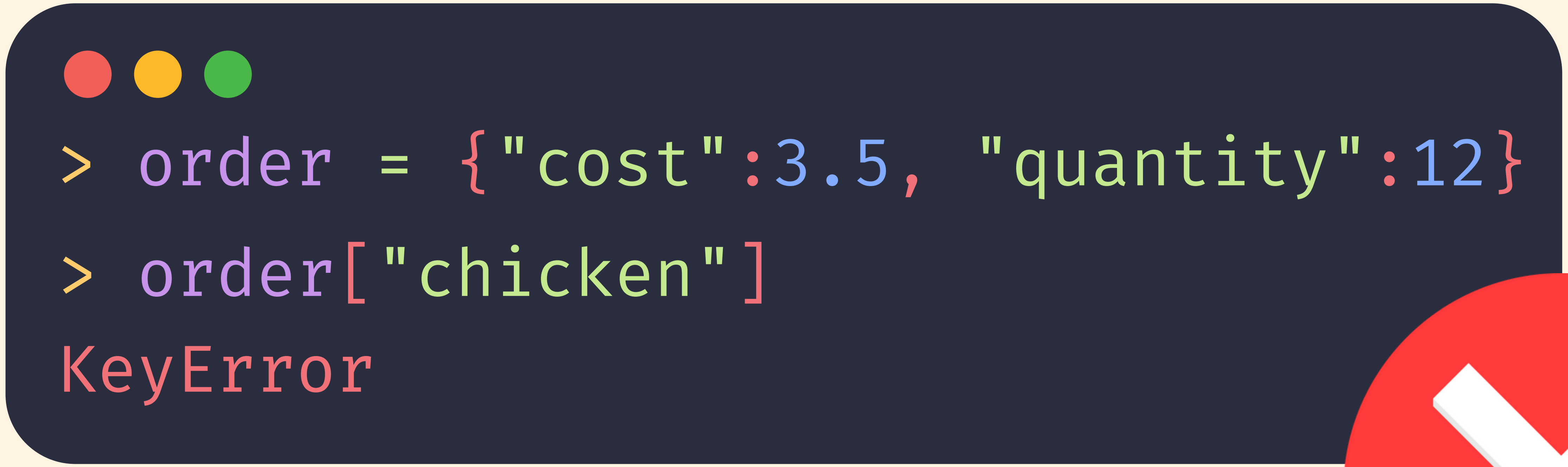
```
> order = {"cost":3.5, "quantity":12}
> order["quantity"]
12
```



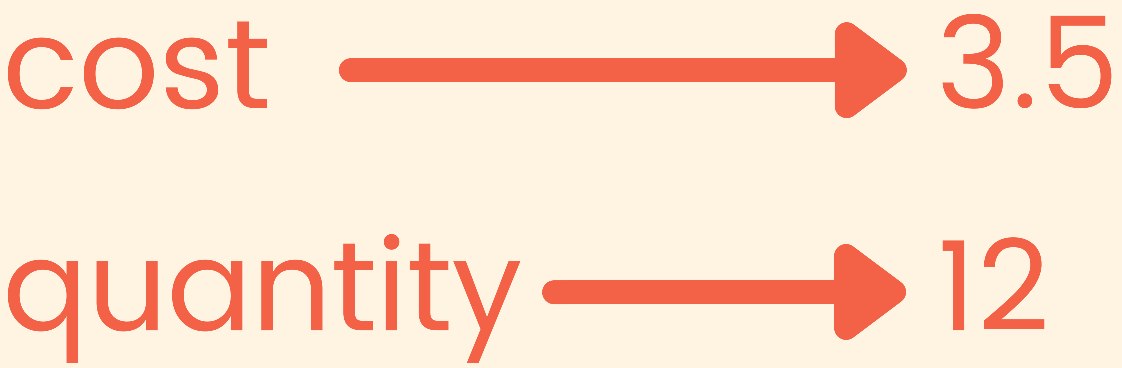
cost → 3.5

quantity → 12

retrieve values using dict[key]



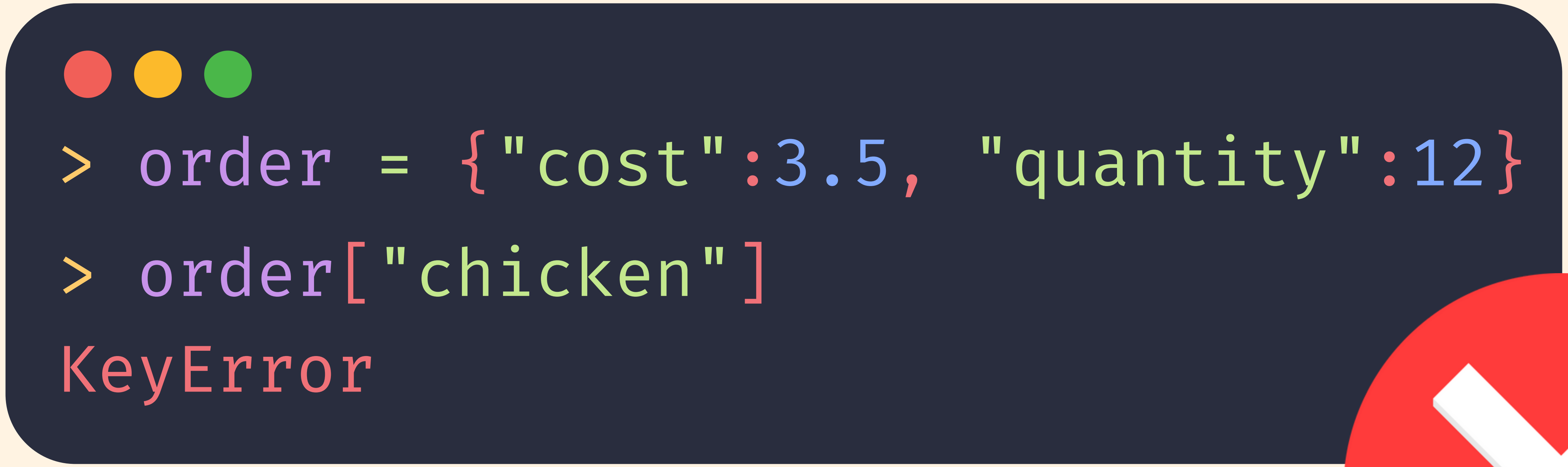
```
> order = {"cost":3.5, "quantity":12}
> order["chicken"]
KeyError
```



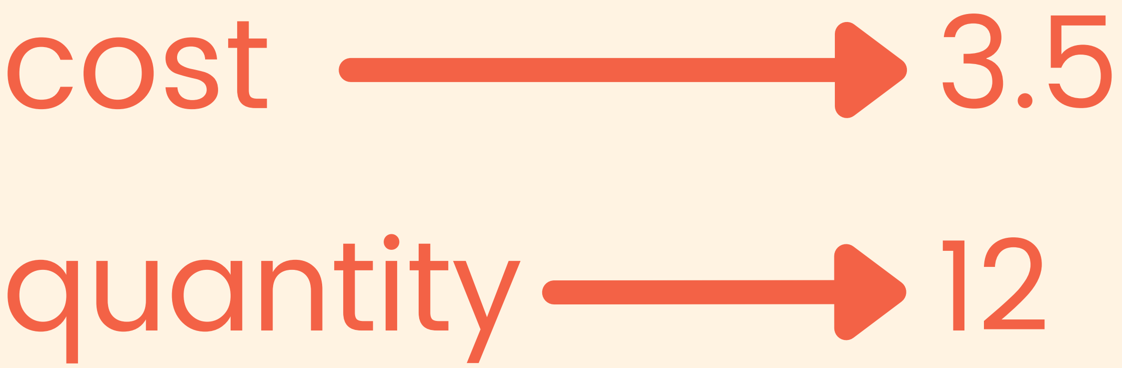
cost → 3.5

quantity → 12

retrieve values using **dict[key]**



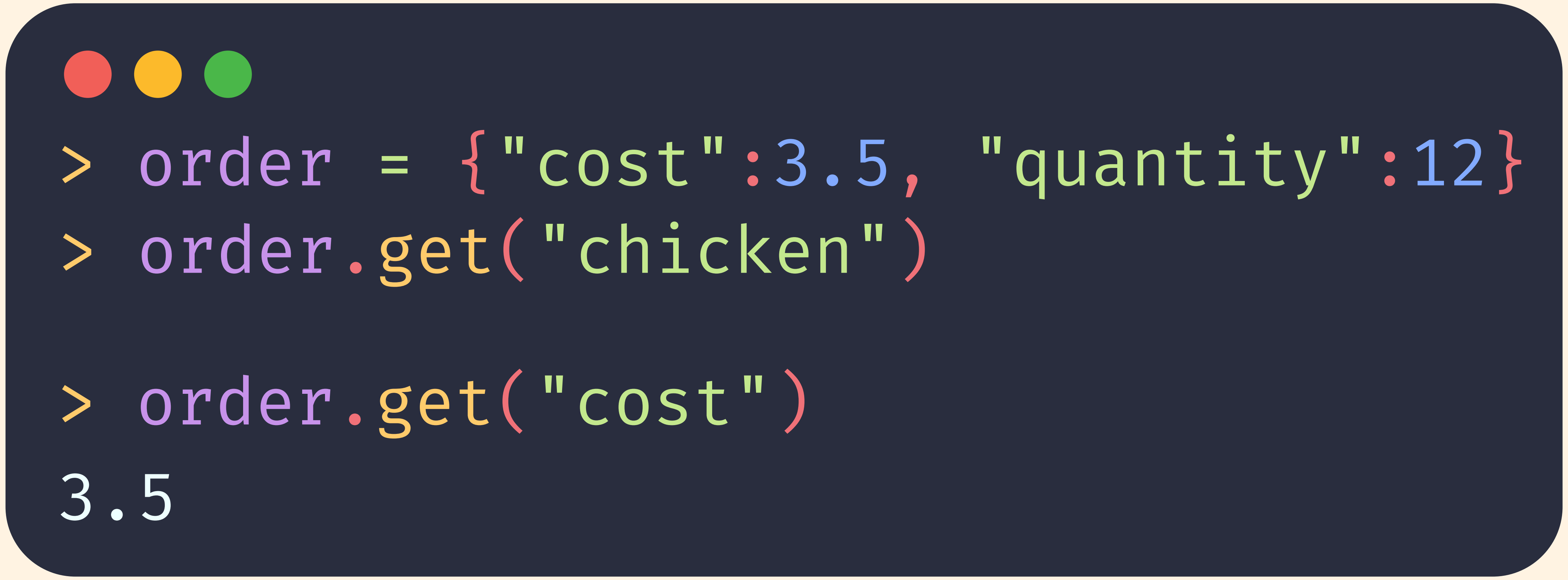
```
> order = {"cost":3.5, "quantity":12}
> order["chicken"]
KeyError
```



cost → 3.5

quantity → 12

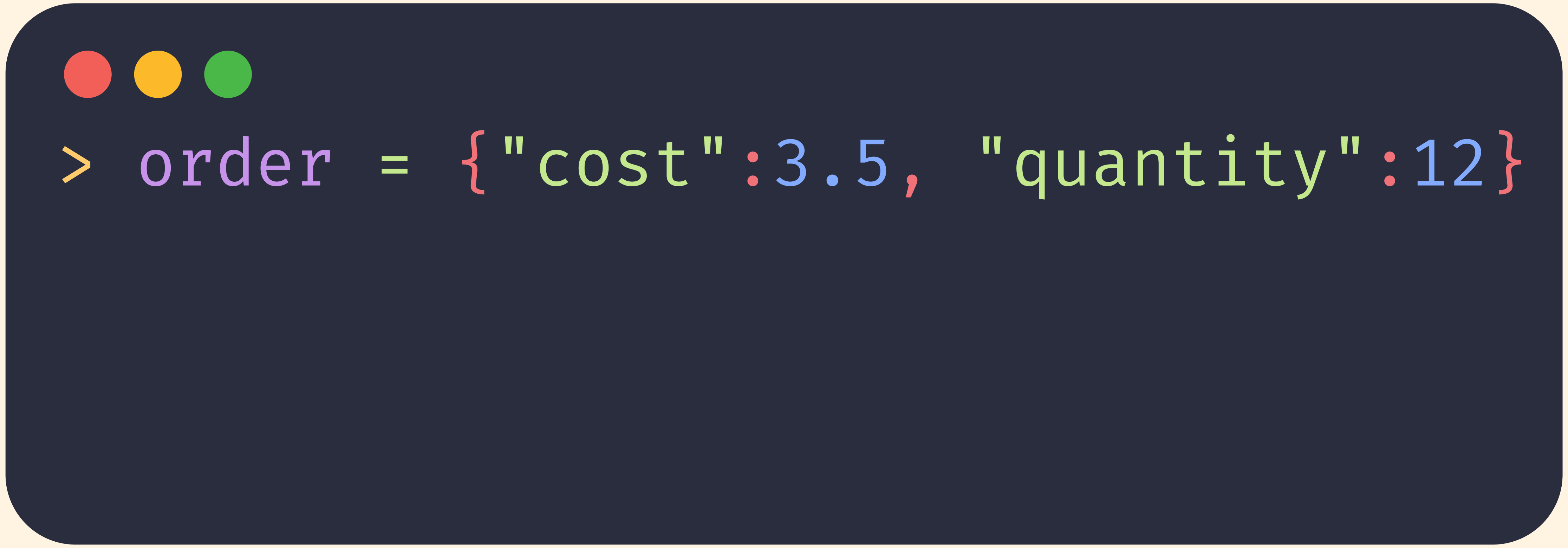
dict.get()



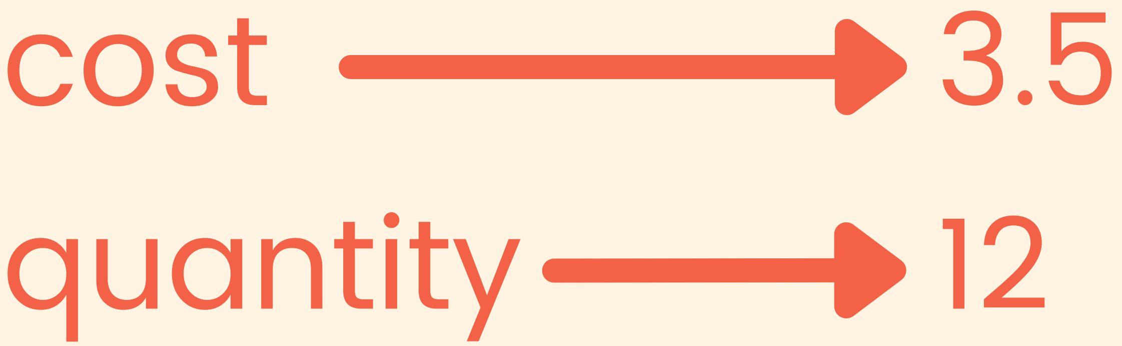
```
> order = {"cost":3.5, "quantity":12}
> order.get("chicken")

> order.get("cost")
3.5
```

The **get()** method will look for a given key in a dictionary. If the key exists, it will return the corresponding value. Otherwise it returns None

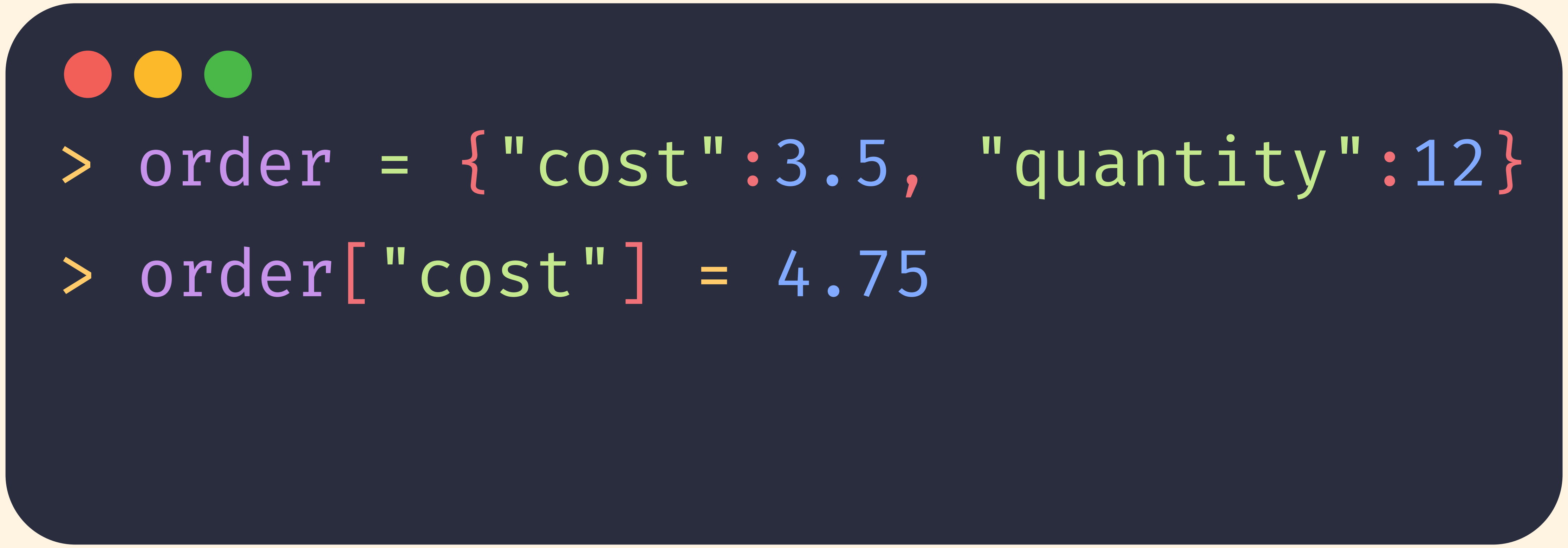
update/add values with dict[key]



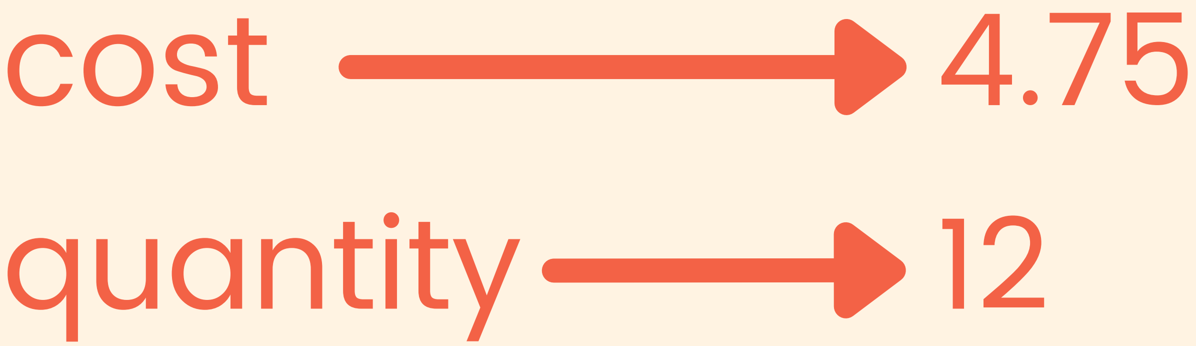
```
> order = {"cost":3.5, "quantity":12}
```



update/add values with dict[key]



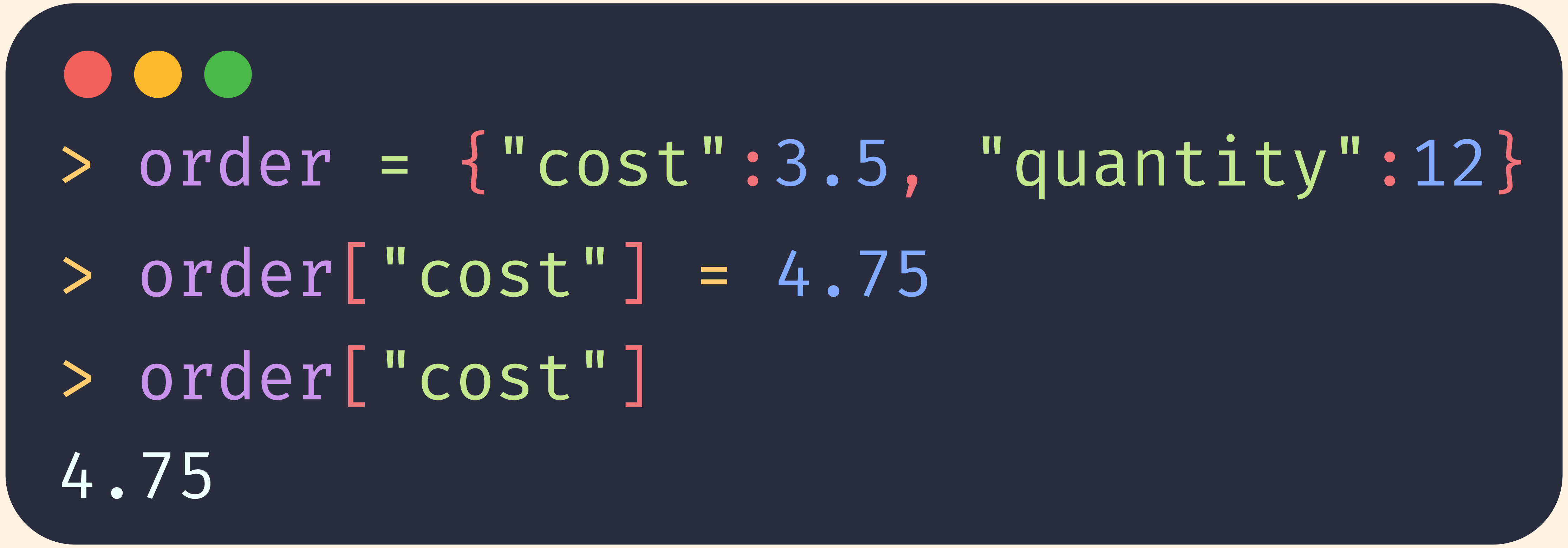
```
> order = {"cost":3.5, "quantity":12}
> order["cost"] = 4.75
```



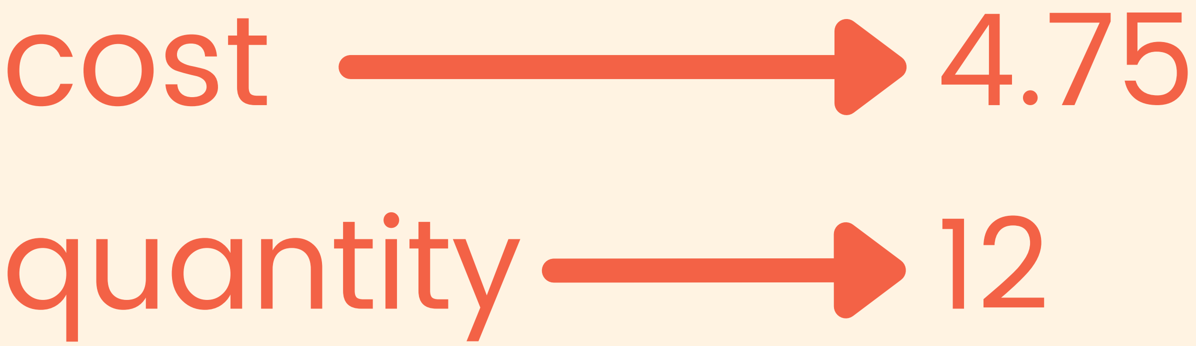
cost → 4.75

quantity → 12

update/add values with dict[key]



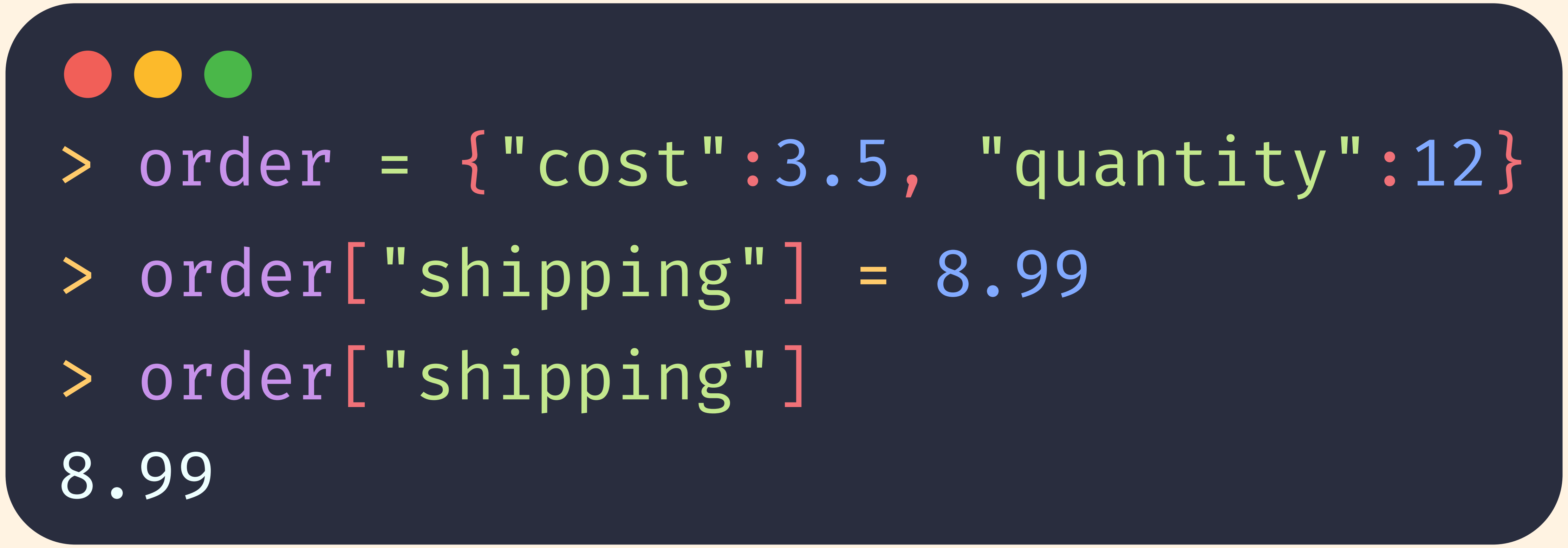
```
> order = {"cost":3.5, "quantity":12}
> order["cost"] = 4.75
> order["cost"]
4.75
```



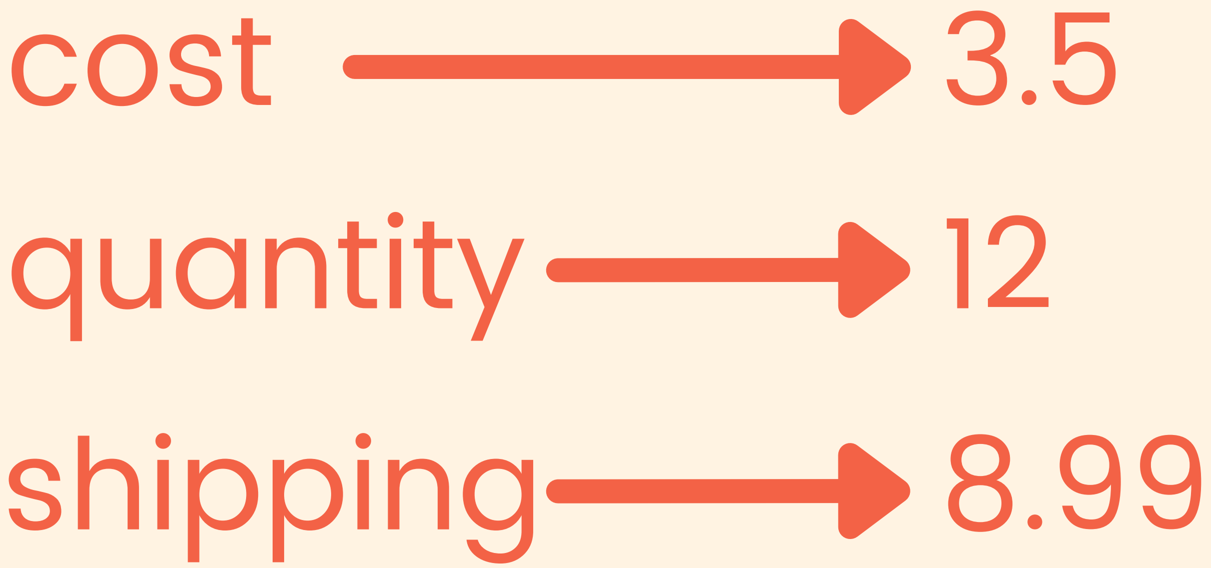
cost → 4.75

quantity → 12

update/add values with dict[key]



```
> order = {"cost":3.5, "quantity":12}
> order["shipping"] = 8.99
> order["shipping"]
8.99
```

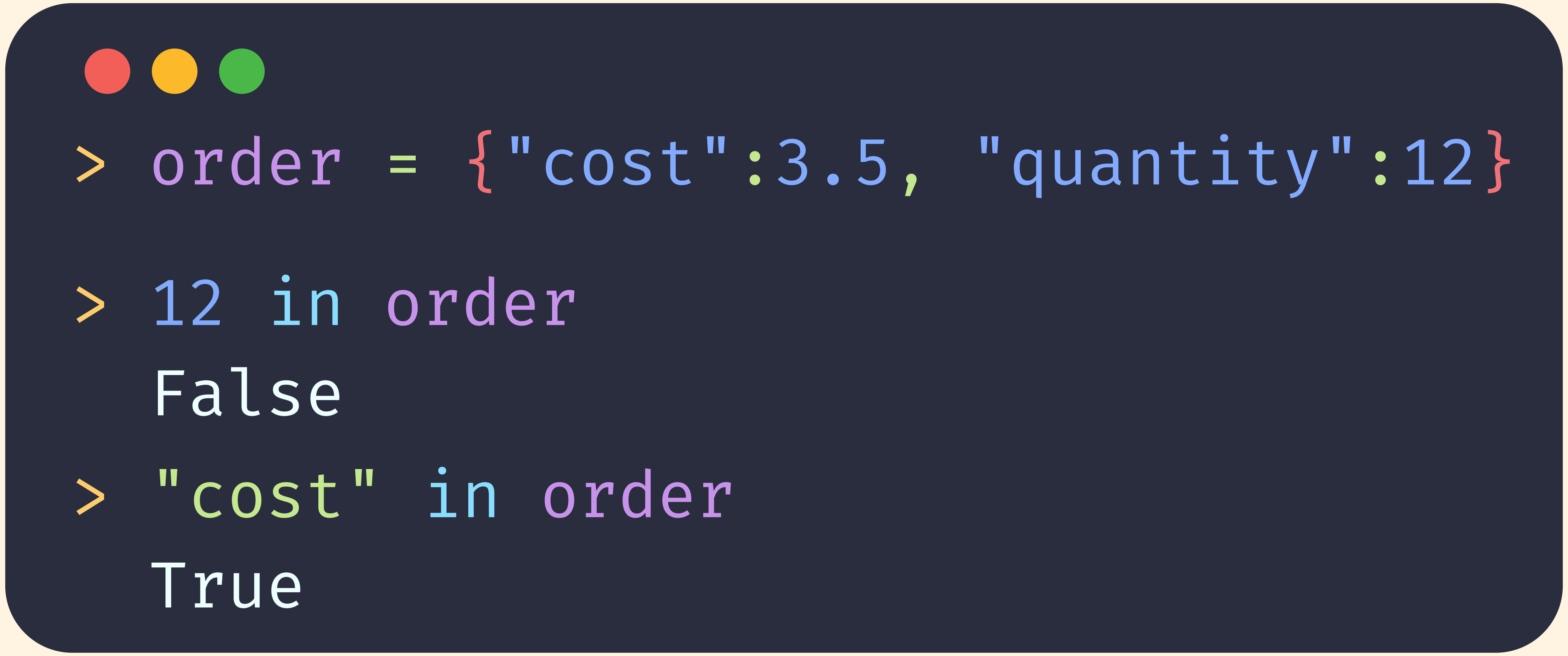


cost → 3.5

quantity → 12

shipping → 8.99

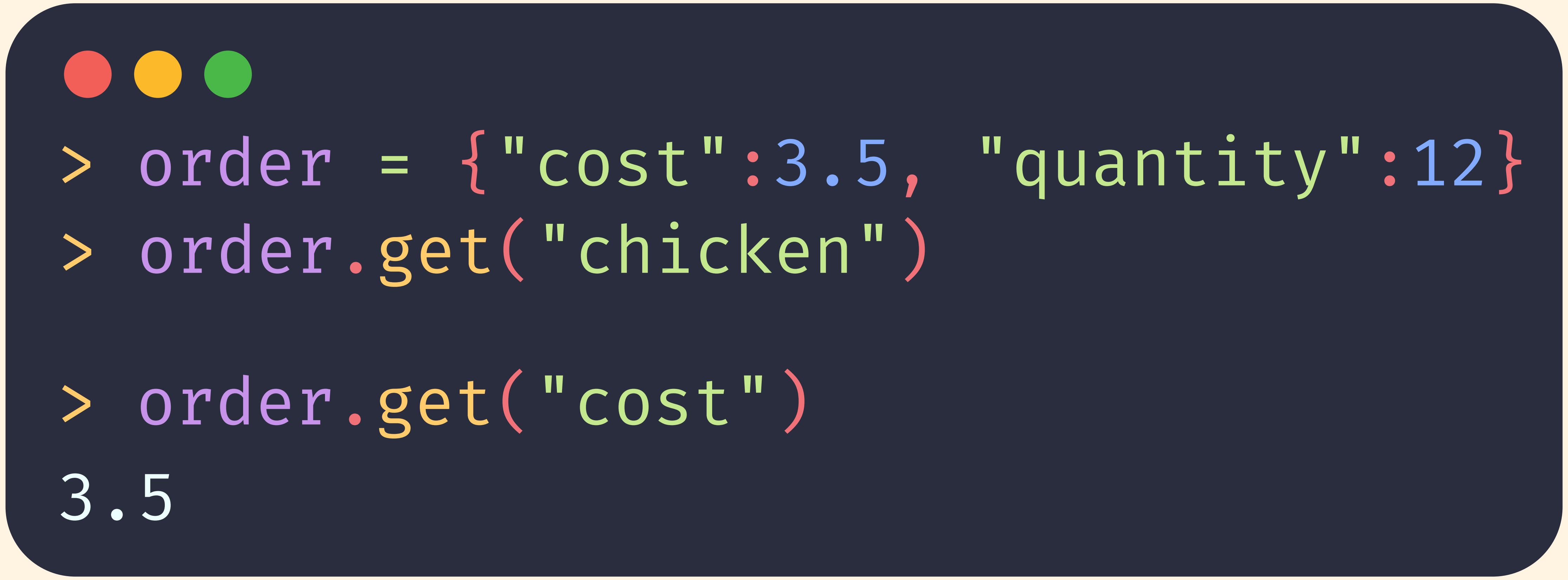
in works with dictionaries too!



```
> order = {"cost":3.5, "quantity":12}

> 12 in order
  False
> "cost" in order
  True
```

It will only look at the keys, not the values

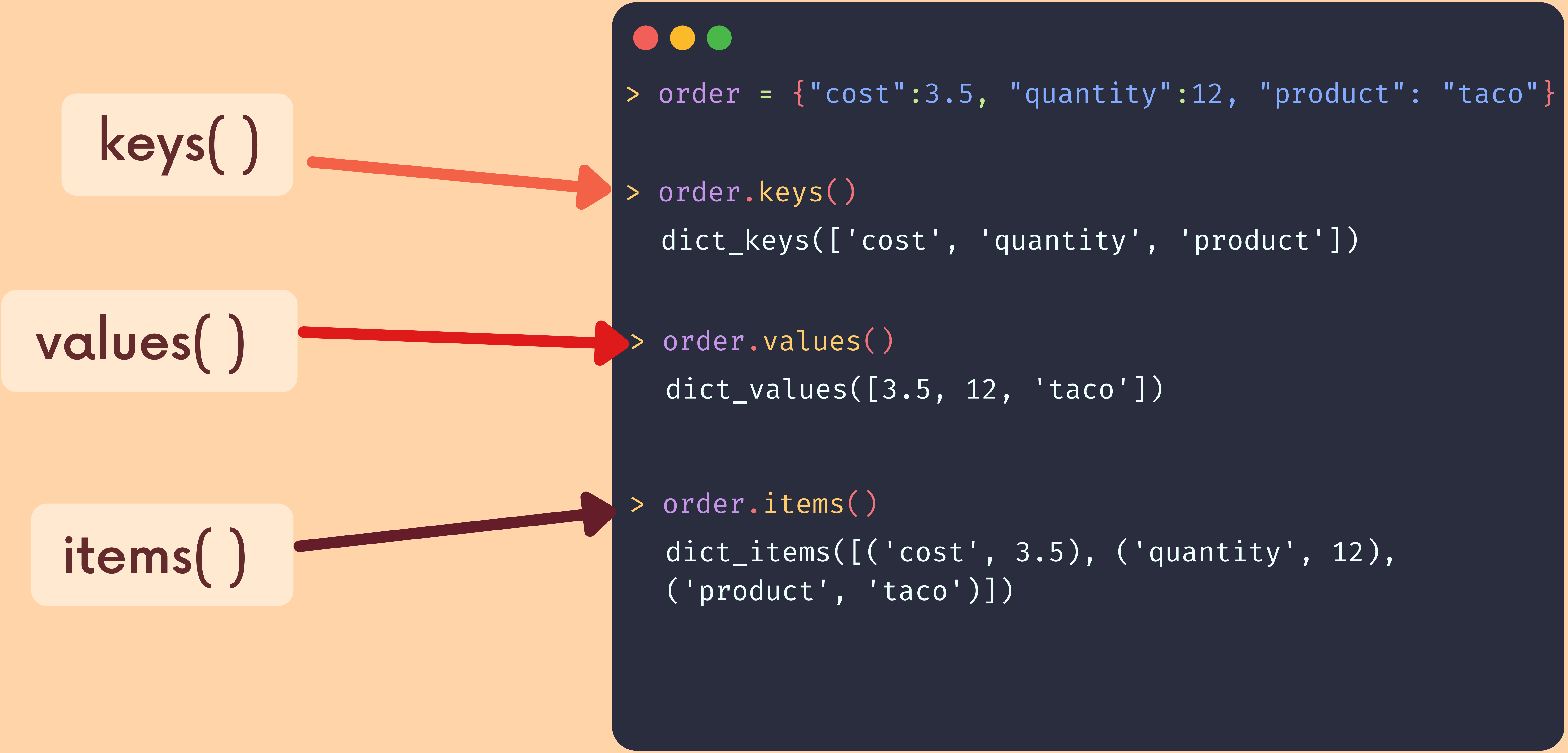
dict.get()



```
> order = {"cost":3.5, "quantity":12}
> order.get("chicken")

> order.get("cost")
3.5
```

The **get()** method will look for a given key in a dictionary. If the key exists, it will return the corresponding value. Otherwise it returns None

.keys, .values, .items



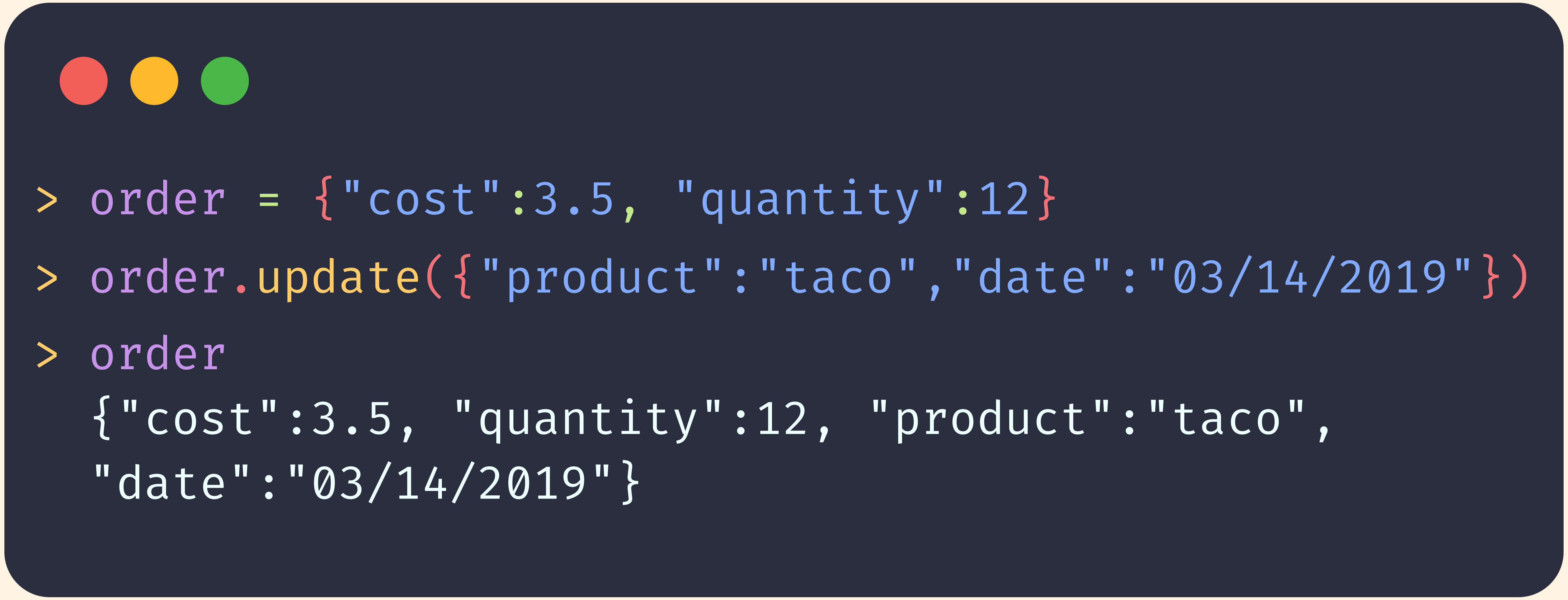
keys( )

values( )

items( )

```
> order = {"cost":3.5, "quantity":12, "product": "taco"}


> order.keys()
  dict_keys(['cost', 'quantity', 'product'])


> order.values()
  dict_values([3.5, 12, 'taco'])


> order.items()
  dict_items([('cost', 3.5), ('quantity', 12),
  ('product', 'taco')])
```

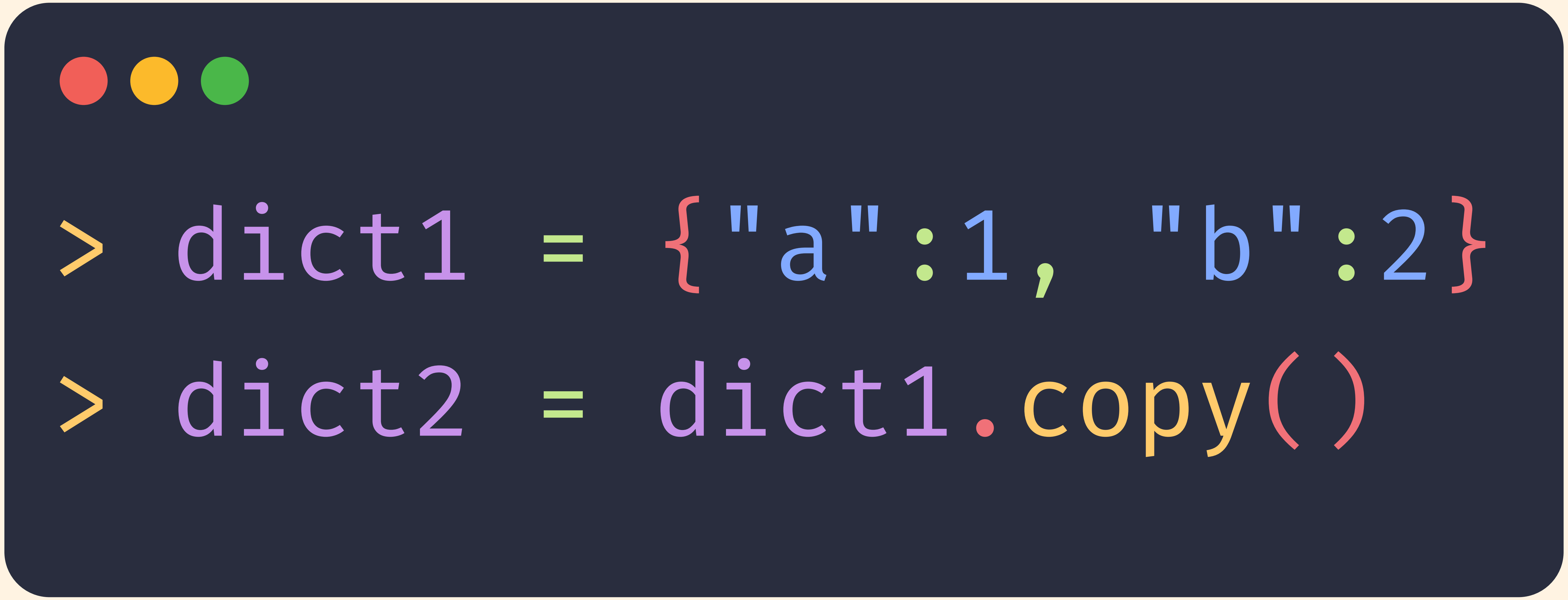
update



```
> order = {"cost":3.5, "quantity":12}
> order.update({"product":"taco","date":"03/14/2019"})
> order
  {"cost":3.5, "quantity":12, "product":"taco",
  "date":"03/14/2019"}
```

The update method will update a dictionary using the key-value pairs from a second dictionary, passed as the argument.

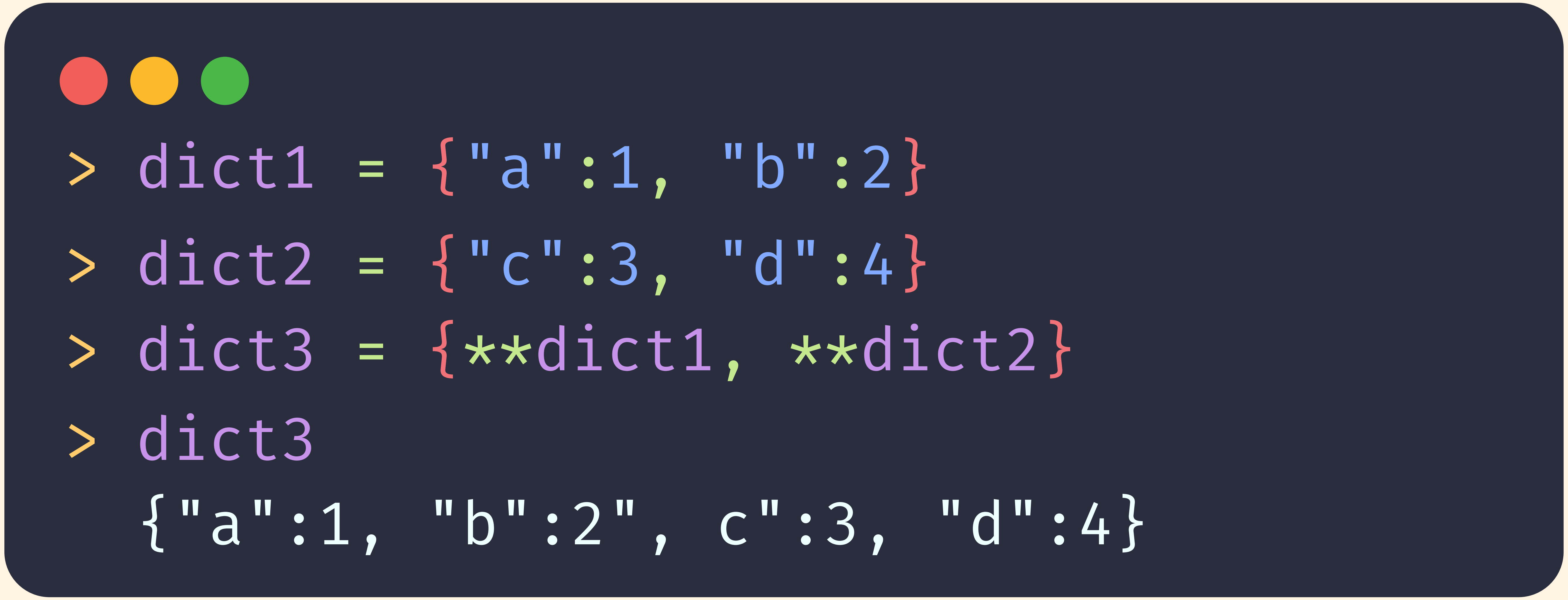
copy



```
> dict1 = {"a":1, "b":2}
> dict2 = dict1.copy()
```

The **copy** method creates and returns a copy of an existing dictionary. It performs a shallow copy.

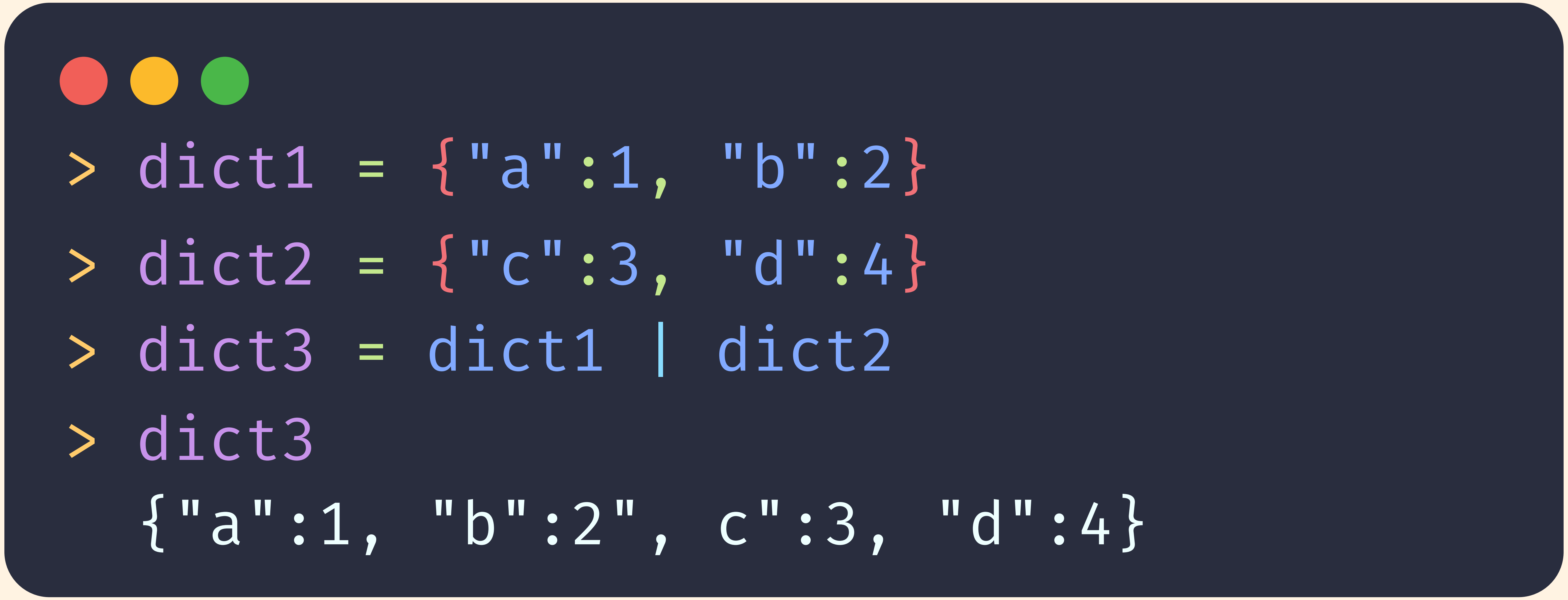
** trick



```
> dict1 = {"a":1, "b":2}
> dict2 = {"c":3, "d":4}
> dict3 = {**dict1, **dict2}
> dict3
  {"a":1, "b":2", c":3, "d":4}
```

We can use two stars ** to combine multiple dictionaries into a new resulting dictionary.

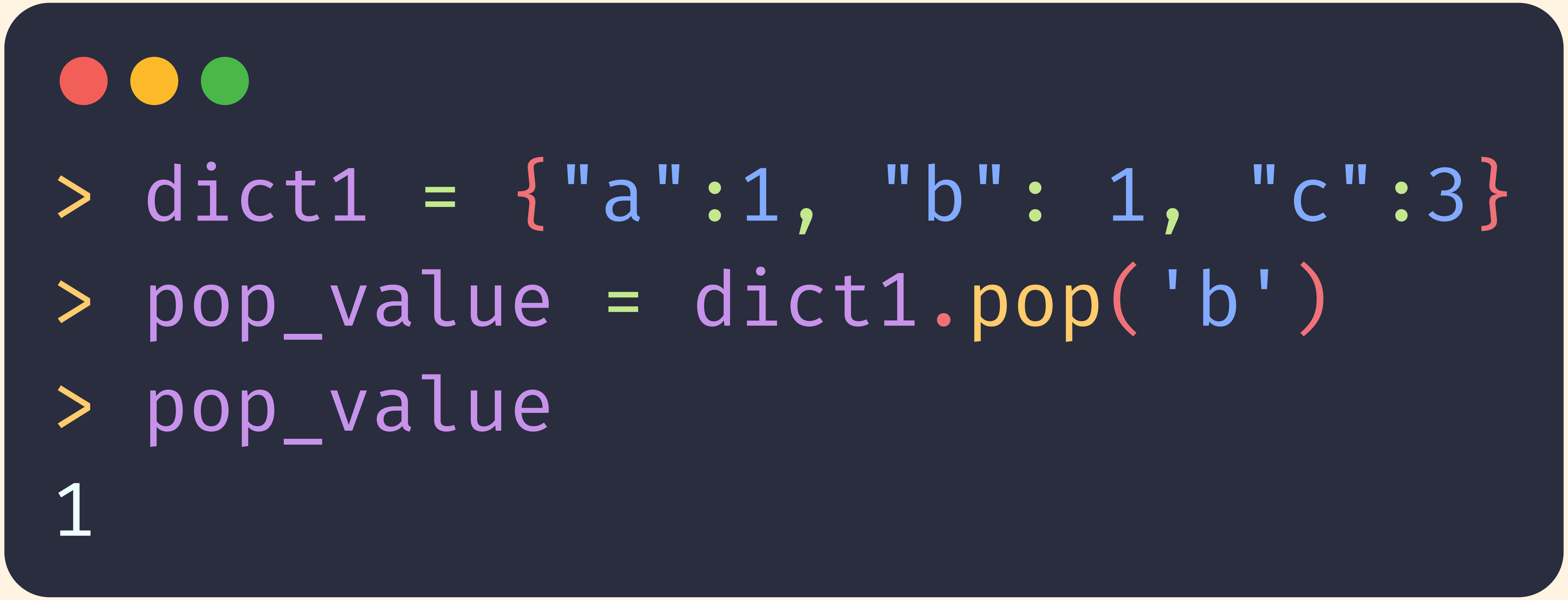
dict union



```
> dict1 = {"a":1, "b":2}
> dict2 = {"c":3, "d":4}
> dict3 = dict1 | dict2
> dict3
  {"a":1, "b":2", c":3, "d":4}
```

Python 3.9 added the dict union operator (|) It will return a new dict containing the items from the left and the right dicts. In the case of duplicated keys, the right side "wins"

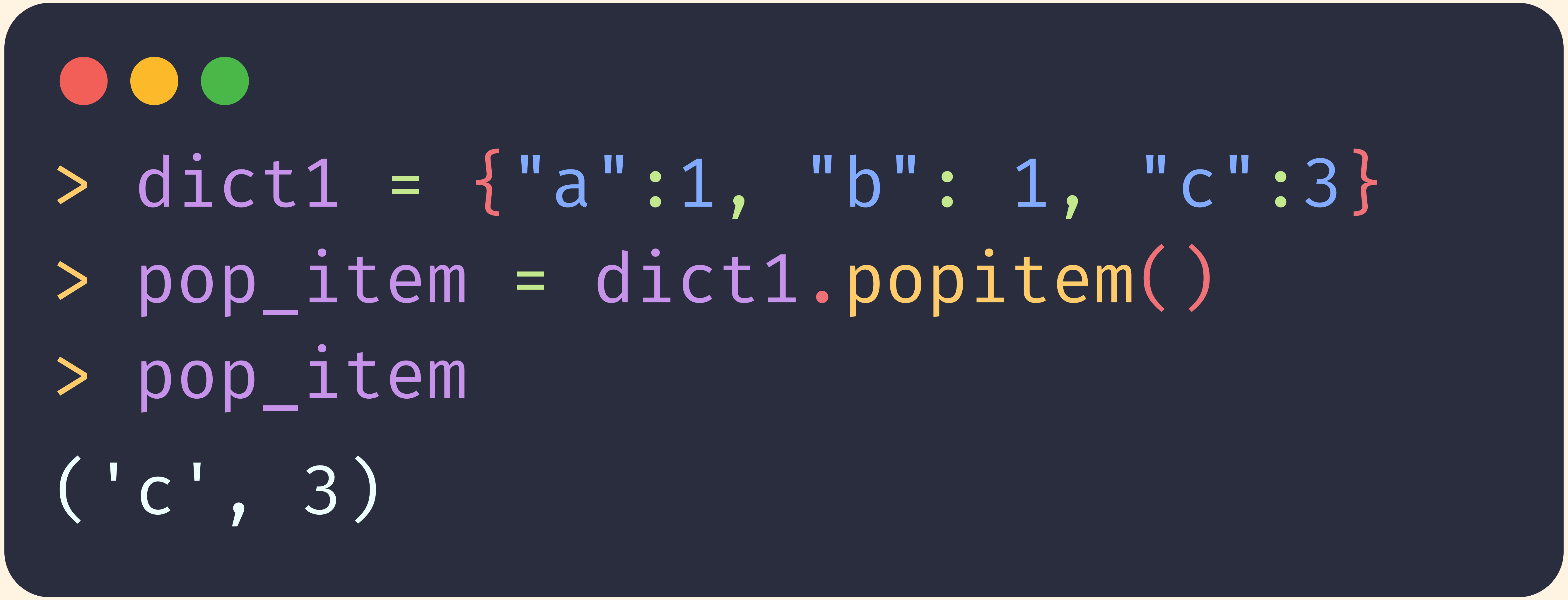
pop



```
> dict1 = {"a":1, "b": 1, "c":3}
> pop_value = dict1.pop('b')
> pop_value
1
```

The pop() method accepts a key and will delete the corresponding key-value pair in the dictionary. It returns the deleted value.

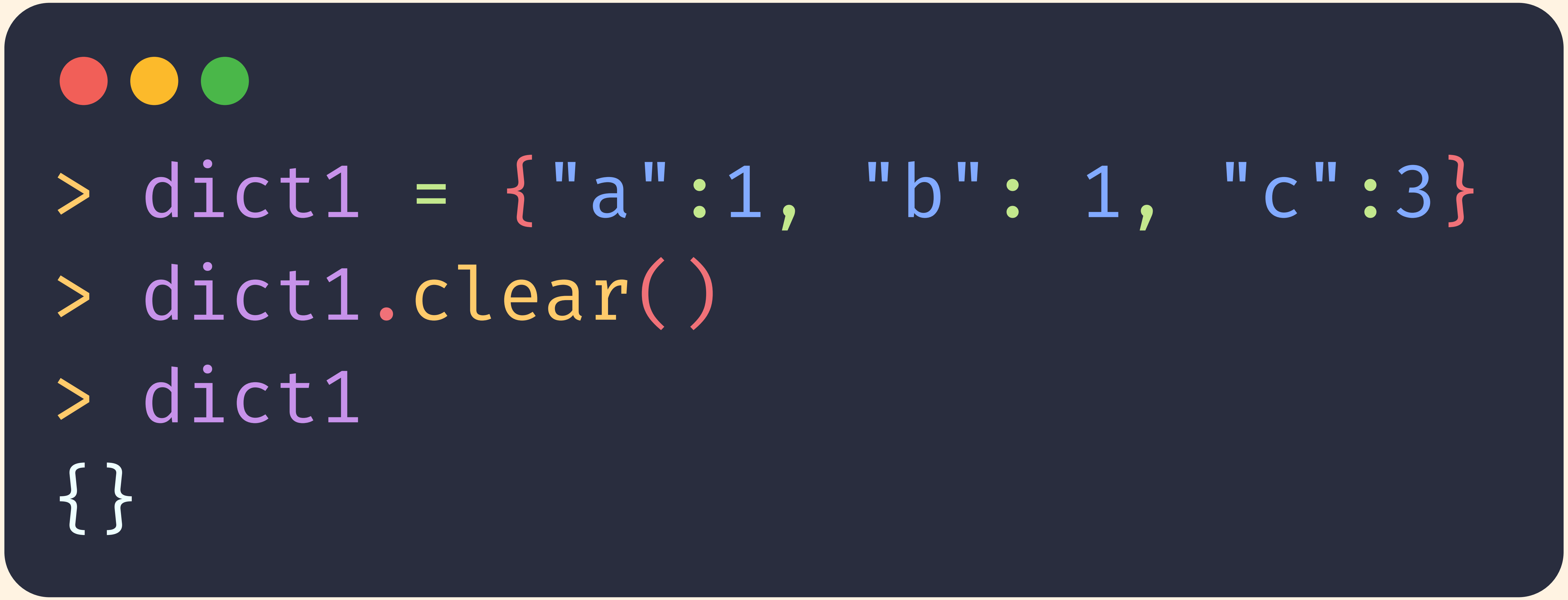
popitem



```
> dict1 = {"a":1, "b": 1, "c":3}
> pop_item = dict1.popitem()
> pop_item
('c', 3)
```

popitem() deletes the most recently added key-value pair. It returns the item as a tuple.

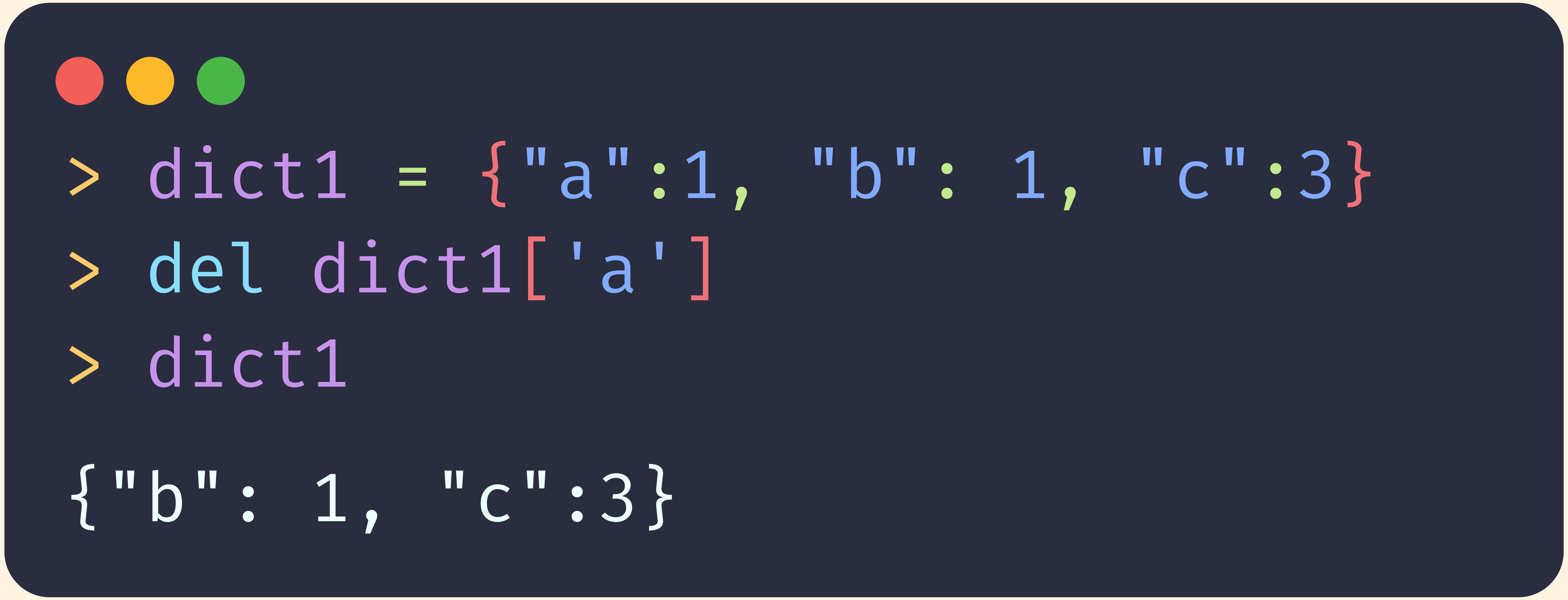
clear



```
> dict1 = {"a":1, "b": 1, "c":3}
> dict1.clear()
> dict1
{}
```

clear() deletes all items from a dictionary.
It returns None.

del



```
> dict1 = {"a":1, "b": 1, "c":3}
> del dict1['a']
> dict1

{"b": 1, "c":3}
```

We can also use the **del statement** to remove items from a dictionary. Remember, it's not a method!