Python Dictionaries Cheat Sheet

Dictionaries

Python dictionaries are an ordered and mutable data type that allow to store data as key-value pairs. That means there are no indices; instead, there are unique keys to access a value stored in dictionaries.

Defining a Dictionary

```
>>> d = {'a': True, 42: 0.3, 0.2: "Hi"}
>>> d
  {'a': True, 42: 0.3, 0.2: 'Hi'}
>>> e = dict(a=0, b=1, c=2, d=3)
>>> e
  {'a': 0, 'b': 1, 'c': 2, 'd': 3}
```

Retrieving Dictionary Elements

```
>>> d = {'a': True, 42: 0.3, 0.2: "Hi"}
>>> l['a']
  True
>>> l[42]
  0.3
>>> l[0.2]
  "Hi"
```

Setting and Removing Dictionary Elements

```
>>> d = {'a': True, 42: 0.3, 0.2: "Hi"}
>>> d['a'] = -12
>>> d
  {'a': -12, 42: 0.3, 0.2: 'Hi'}
>>> d[False] = 'b'
>>> d
  {'a': -12, 42: 0.3, 0.2: 'Hi', False: 'b'}
>>> del d['a']
>>> d
  {42: 0.3, 0.2: 'Hi', False: 'b'}
```

dict.fromkeys()

Creates a dictionary from a list of keys with a default value.

Syntax

```
dict.fromkeys(iterable, value=None)
```

Parameters

`iterable`: iterable containing the keys
`value`: default value

Return Value

dictionary with keys from `iterable` and all values set to `value`

```
>>> k = ['a','b','c']
>>> dict.fromkeys(k)
  {'a': None, 'b': None, 'c': None}
>>> dict.fromkeys(k, 0)
  {'a': 0, 'b': 0, 'c': 0}
```

.get()

Returns the value for the given key if it is in the dictionary; otherwise, it returns a default value.

Syntax

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

Parameters

`key`: key for which the value is returned
`default`: value that is returned if key is not in dictionary

Return Value

value for key or default

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d.get('a')
  0
>>> d.get('z')
  None
>>> d.get('z', -1)
  -1
```

.update()

Updates and extends key-value pairs of a dictionary.

Syntax

```
dict.update(dict/iterable)
```

Parameters

`dict/iterable`: dictionary or iterable of tuples to extend and update the dictionary

Return Value

None

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d.update({'a':0})
>>> d
  {'a': 0, 'b': 2, 'c': 3}
>>> d.update({'d':4})
>>> d
  {'a': 0, 'b': 2, 'c': 3, 'd': 4}
>>> d.update([('e',5)])
>>> d
  {'a': 0, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Dictionaries from Lists

Two lists of equal length can be turned into a dictionary where one list represents the keys and the other list represents the values. If the lists are of uneven length, the longer list is truncated.

Syntax

```
>>> k = ['a','b','c']
>>> v = [0, 1, 2, 3] ← truncated
>>> d = dict(zip(k, v))
>>> d
  {'a': 0, 'b': 1, 'c': 2}
```

.pop()

Removes a key-value pair from a dictionary and returns the value.

Syntax

```
dict.pop(key)
```

Parameters

`key`: key of the key value pair that is removed

Return Value

value

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d.pop('a')
  0
>>> d
  {'b': 1, 'c': 2, 'd': 3}
```

.popitem()

Removes an arbitrary key-value pair from a dictionary.

Syntax

```
dict.popitem()
```

Parameters

None

Return Value

tuple containing the key value pair

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d.popitem()
  ('d', 3)
>>> d
  {'a': 0, 'b': 1, 'c': 2}
```

.clear()

Removes all key-value pairs from a dictionary.

Syntax

```
dict.clear()
```

Parameters

None

Return Value

None

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d.clear()
>>> d
  {}
```

.values()

Returns all values of dictionary as an iterable.

Syntax

```
dict.values()
```

Parameters

None

Return Value

iterable containing all values of the dictionary

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d.values()
  dict_values([0, 1, 2, 3])
```

.keys()

Returns all keys of dictionary as an iterable.

Syntax

```
dict.keys()
```

Parameters

None

Return Value

iterable containing all keys of the dictionary

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d.keys()
  dict_keys(['a', 'b', 'c', 'd'])
```

Dictionaries in if,elif,else

The `in` operator can be used to check if a key, value, or a key-value pair is in a dictionary.

Syntax

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> if 'z' in d.keys():
...     print("found z")
... elif 4 in d.values():
...     print("found 4")
... elif ('a',0) in d.items():
...     print("found 'a':0")
... else:
...     print("did not find anything")
  found 'a':0
```

Nested Dictionaries

Dictionaries can be nested to create complex data structures.

Syntax

```
>>> d = {'a': {0: False, 1: True },
         'b': {0: -0.5,  1: 0.5 }}
>>> d['a']
  {0: False, 1: True}
>>> d['b']
  {0: -0.5, 1: 0.5}
>>> d['a'][1]
  True
>>> d['b'][0]
  -0.5
```

Merging Dictionaries |

The pipe operator | merges two dictionaries.

Syntax

```
dict | dict
```

Parameters

`dict`: dictionaries that are merged

Return Value

merged dictionaries

```
>>> d = {'a': 0, 'b': 1}
>>> e = {'c': 2, 'd': 3}
>>> f = {'a': 4, 'd': 5}
>>> d | e
{'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> d | f
{'a': 4, 'b': 1, 'd': 5}
>>> f | d
{'a': 0, 'd': 5, 'b': 1}
>>> d | e | f
{'a': 4, 'b': 1, 'c': 2, 'd': 5}
```

.copy()

Returns a copy of a dictionary.

Syntax

```
dict.copy()
```

Parameters

None

Return Value

copy of the dictionary

Copy vs. Reference

A reference to an object is created with the = operator. When the `.copy()` functions is callend on an object the object is duplicated and returned.

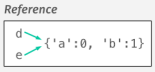
Multiple variables reference a single object:

```
>>> d = {'a': 0, 'b': 1}
>>> e = d
>>> e
  {'a': 0, 'b': 1}
>>> d['a'] = -1
>>> d
  {'a': -1, 'b': 1}
>>> e
  {'a': -1, 'b': 1}
```

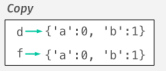
An object copy is created and assigned to a new variable:

```
>>> d = {'a': 0, 'b': 1}
>>> f = d.copy()
>>> f
  {'a': 0, 'b': 1}
>>> d['a'] = -1
>>> d
  {'a': -1, 'b': 1}
>>> f
  {'a': 0, 'b': 1}
```

Reference



Copy



Dictionaries in Loops

The `in` operator can be used to loop over each key, value, or both in a dictionary.

for loop over keys

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> for k in d:
...     print(k, end=',')
  a,b,c,d,
>>> for k in d.keys():
...     print(k, end=',')
  a,b,c,d,
```

for loop over values

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> for v in d.values():
...     print(v, end=',')
  0,1,2,3,
```

for loop over key value pairs

```
>>> d = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
>>> for k, v in d.items():
...     print(f"{k}:{v}", end=',')
  a:0,b:1,c:2,d:3,
```

.items()

Returns all key-value pairs of as tuples in an iterable.

Syntax

```
dict.items()
```

Parameters

None

Return Value

iterable containing all key value pairs as tuples

```
>>> d = {'a': 0, 'b': 1, 'c': 2}
>>> d.items()
  dict_items([('a',0),('b',1),('c',2)])
```

.setdefault()

Adds a key with default value as a key value pair to a dictionary if the key is not in the dictionary. If the key is in the dictionary, the value is updated.

Syntax

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

Parameters

`key`: key that is added to the dictionary
`default`: value that is added to the dictionary for key

Return Value

The value the was added to the dictionary otherwise None

```
>>> d = {'a': 0, 'b': 1}
>>> d.setdefault('d')
>>> d
  {'a': 0, 'b': 1, 'd': None}
>>> d.setdefault('d', 2)
>>> d
  {'a': 0, 'b': 1, 'd': 2}
>>> d.setdefault('e', 3)
  3
```

max() & min()

Returns the maximum/minimum element of an iterable.

Syntax

```
max(iterable, key=None)
min(iterable, key=None)
```

Parameters

`iterable`: list (iterable) that is sorted
`key`: sorting key

Return Value

maximum/minimum element

Maximum/Minimum Key

```
>>> d = {'a': 3, 'b': 2, 'c': 1, 'd': 0}
>>> max(d) ... >>> min(d)
  'd'       ... >>> 'a'
```

Maximum/Minimum Value

```
>>> d = {'a': 3, 'b': 2, 'c': 1, 'd': 0}
>>> max(d.values()) ... >>> min(d.values())
    3              ... >>> 0
```

Get Key for Maximum/Minimum Value

```
>>> d = {'a': 3, 'b': 2, 'c': 1, 'd': 0}
>>> max(d,key=d.get) .. >>> min(d,key=d.get)
    'a'               .. >>> 'd'
```


k0nze builds
k0nze_
discord.k0nze.dev
patreon.com/k0nze

www.k0nze.dev