Python Dictionaries Cheat Sheet

Dictionaries in Python

A dictionary is an unordered collection of key: value pairs

Introduction to dictionaries

```
# Creating dictionaries
dict1 = {}         # empty dictionary
dict2 = dict()     # empty dictionary

# Keys must be immutable types, values can any type
# invalid_dict = {[1,2,3]: "123"}    # => Raises a TypeError: unhashable type: 'list'

# key of type tuple is permitted (immutable). Values can be of any type
valid_dict = {(1,2,3):[1,2,3], 3: 'abc', 'abc':{14,'a'}, 4.5:True}

product = {'id':1234, 'name':'Laptop'}
product['seller'] = 'Amazon'           # adds a new key:value pair
product.update({'price':888.99})    # another way to add to a dictionary

p = product['price']              # getting the value of a key,  p is 888.00

del product['seller']            # removing a key:value pair
print(product)                    # => {'id': 1234, 'name': 'Laptop', 'price': 888.99}
#seller = product['seller']    # Looking up a non-existing key is a KeyError

# dict.get() retrieves the value of a key
product.get("id")          # => 1234
product.get("review")    # => None -> it doesn't return KeyError

# dict.get() supports a default argument which is returned when the key is missing
product.get("id", 4)               # => 1234
product.get("review", 'N/A')    # => 'N/A'
```

```
# dict.pop() removes the specified key and returns the corresponding value.
# If key is not found, a default value is given, otherwise KeyError is raised
name = product.pop('name')                  # name is 'Laptop'
print(product)                              # => {'id': 1234, 'price': 888.99}
# name = product.pop('name')                # => KeyError: 'name', key 'name' doesn't exist anymore
name = product.pop('name', 'No such key')   # => name is 'No such key'
```

Dictionary Operations and Methods

```
product = {'id':1234, 'price':888.99}

# in and not in operators test dictionary key membership
'price' in product  # => True
5 in product        # => False

# Getting dictionary views: dict.keys(), dict.values() and dict.items()
keys = product.keys()    # getting all keys as an iterable
keys = list(keys)        # it can be converted to a list
print(keys)              # => ['id', 'price']

values = product.values()    # getting all values as an iterable
values = list(values)        # it can be converted to a list
print(values)                # => [1234, 888.99]

key_values = product.items()    # getting all key:value pairs as tuples
key_values = list(key_values)   # it can be converted to a list of tuples
print(key_values)               # => [('id', 1234), ('price', 888.99)]

# dict.copy() creates a copy of a dictionary
prod = product.copy()

# Return the number of key:value pairs in the dictionary
len(prod)

#  dict.clear() removes all items from dictionary
product.clear()
```

Iterating over a dictionary

```
product = {'id':1234, 'price':888.99}

# Iterating over the keys
for k in product:
   print(f'key:{k}')
```

```
#equivalent to:
for k in product.keys():
   print(f'key:{k}')

# Iterating over the values
for v in product.values():
   print(f'value:{v}')

# Iterating over both the keys and the values
for k,v in product.items():
   print(f'key:{k}, value:{v}')
```

zip() Built-in Function

Returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables.

The iterator stops when the shortest input iterable is exhausted.

```
years = [2015, 2016, 2017, 2018]
sales = [20000, 30000, 40000, 45000]

# Zipping in a list of tuples
sales_list = list(zip(years, sales))
print(sales_list)   # => [(2015, 20000), (2016, 30000), (2017, 40000), (2018, 45000)]

# Zipping in a dictionary
sales_dict = dict(zip(years, sales))
print(sales_dict)   # => {2015: 20000, 2016: 30000, 2017: 40000, 2018: 45000}
```

Dictionary Comprehension

```
d1 = {'a':1, 'b':2, 'c':3}

## Doubled values
d2 = {k: v * 2 for k, v in d1.items()}
print(d2)    # => {'a': 2, 'b': 4, 'c': 6}

## Doubled keys, squared values
d3 = {k * 2: v * 3 for k, v in d1.items() }
print(d3)    # => {'aa': 3, 'bb': 6, 'cc': 9}
```

```
sales = {2015: 20000, 2016: 30000, 2017: 40000, 2018: 45000}
# Creating a dictionary called vat considering that the value added tax is 20%
vat = {k: v * 0.2 for k, v in sales.items()}
print(vat)    # => {2015: 4000.0, 2016: 6000.0, 2017: 8000.0, 2018: 9000.0}
```