# Dictionaries

Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is **ordered, changeable and does not allow duplicates(keys).**

Dictionaries are written with curly brackets.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

The values in dictionary items can be of any data type.

# Creating a dictionary

1. It consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value. Example: Name > Phone number
2. You can define a dictionary by enclosing a comma-separated list of key-value pairs in curly braces ({}). A colon (:) separates each key from its associated value:

d={12 : "Ram", 13: "Shyam", 14: "Ram"}

**Syntax:**

1. class dict(** kwarg)

The special syntax **kwargs in function definitions in python is used to pass a keyworded, variable-length argument list.

2. **class dict(mapping, **kwarg)**
3. **class dict(iterable, **kwarg)**

Return a new dictionary initialized from an optional positional argument and a possibly empty set of keyword arguments.

In [1]:
```
# 1:
a = dict(one=1, two=2, three=3)
print(a)

# 2:
b = {'one': 1, 'two': 2, 'three': 3}
print(b)

# 3:
c = dict({'three': 3, 'one': 1, 'two': 2})
print(c)

# 4: using zip() function
d = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
print(d)

# 5
e = dict([('one', 1), ('two', 2), ('three', 3)])
print(e)

print(a == b == c == d == e)
```

```
{'one': 1, 'two': 2, 'three': 3}
{'one': 1, 'two': 2, 'three': 3}
{'three': 3, 'one': 1, 'two': 2}
{'one': 1, 'two': 2, 'three': 3}
{'one': 1, 'two': 2, 'three': 3}
True
```

In [2]:
```
# creating empty dictionary
mydict = {}
mydict
```

Out[2]:
```
{}
```

In [3]:
```
a = dict(one=1, two=2, three=3); a
```

Out[3]:
```
{'one': 1, 'two': 2, 'three': 3}
```

# How dictionaries are stored in memory

1. Python dictionaries are implemented as hash tables.
2. Python hash table is just a contiguous block of memory
3. Each entry in the table is actually a combination of the three values: < hash, key, value >. This is implemented as a C struct.
4. Hashing is a concept in computer science which is used to create high performance, pseudo random access data structures where large amount of data is to be stored and accessed quickly.
5. Dictionaries work by computing a hash code for each key stored in the dictionary using the built-in hash function.
6. The hash code varies widely depending on the key; for example, “Python” hashes to -539294296 while “python”, a string that differs by a single bit, hashes to 1142331976.
7. The hash code is then used to calculate a location in an internal array where the value will be stored.
8. Assuming that you’re storing keys that all have different hash values, this means that dictionaries take constant time — O(1), in computer science notation — to retrieve a key.
9. No sorted order of the keys is maintained, and traversing the internal array as the dict.keys and dict.items methods do will output the dictionary’s content in some arbitrary jumbled order.

In [4]:
```
a = dict(one=1, two=2, three=3); a
```

Out[4]:
```
{'one': 1, 'two': 2, 'three': 3}
```

In [5]:
```
hash('one'), hash('two'), hash('three')
```

Out[5]:
```
(-7930654172195661816, -6692945843925490994, -5510635465417144172)
```

In [6]:
```
# Demonstrating hash function
a = 3931459000720265852 % 3
b = 4715497456990954285 % 3
a, b
```

Out[6]:
```
(2, 1)
```

A given key can appear in a dictionary only once. Duplicate keys are not allowed. A dictionary maps each key to a corresponding value, so it doesn’t make sense to map a particular key more than once. If you specify a key a second time during the initial creation of a dictionary, then the second occurrence will override the first.

In [7]:
```
# Duplicate values with same key
a= {'one': 1, 'two': 2, 'three':3, 'one': 2}
a
```

Out[7]:
```
{'one': 2, 'two': 2, 'three': 3}
```

If we have to search an element in list, it will traverse the entire list. He worst case time complexity becomes O(n) while in dictionary, it visits the index value using the hash code in the hash table and sees if the key is present or not. So the time complexity reduces to O(1).

In [8]:
```
a= [1,3,2,4]
4 in a
```

Out[8]:
```
True
```

In [9]:
```
d= {'one': 1, 'two': 2, 'three':3}
'one' in d
```

Out[9]:
```
True
```

# Access items in dictionary

You can start by creating an empty dictionary, which is specified by empty curly braces. Then you can add new keys and values one at a time. Once the dictionary is created in this way, its values are accessed the same way as any other dictionary.

In [10]:
```
mydict = {}
```

In [11]:
```
# Using keys
thisdict = {
                "brand": "Ford",
                "model": "Mustang",
                "year": 1964
            }
```

In [12]:
```
x = thisdict["model"]
print(x)
print(type(x))
```

```
Mustang
<class 'str'>
```

In [13]:
```
# Using get() method
x = thisdict.get("brand")
print(x)
```

```
Ford
```

In [14]:
```
# The keys() method will return a list of all the keys in the dictionary.
# The list of the keys is a view of the dictionary, meaning that any changes done to the diction
# will be reflected in the keys list.
x = thisdict.keys()
print(x)
```

```
dict_keys(['brand', 'model', 'year'])
```

In [15]:
```
# The values() method will return a list of all the values in the dictionary.
x = thisdict.values()
print(x)
```

```
dict_values(['Ford', 'Mustang', 1964])
```

In [16]:
```
# The items() method will return each item in a dictionary, as tuples in a list.
x = thisdict.items()
print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

In [17]:
```
# To determine if a specified key is present in a dictionary use the in/not in keyword:
'year' in thisdict
# 'brand' not in thisdict
# 'model' in thisdict.keys()
# 'Ford' in thisdict.values()
# 2001 in thisdict.values()
```

Out[17]:
```
True
```

# Loop through a dictionary

In [18]:
```
thisdict = {
               "brand": "Ford",
               "model": "Mustang",
               "year": 1964
            }
```

In [19]:
```
# Print all key names in the dictionary
for i in thisdict:
    print(i)
```

```
brand
model
year
```

In [20]:
```
# Print all values in the dictionary
for i in thisdict:
    print(thisdict[i])
```

```
Ford
Mustang
1964
```

In [21]:
```
# Print all keys and values in the dictionary
for i in thisdict:
    print(i, thisdict[i])
```

```
brand Ford
model Mustang
year 1964
```

In [22]:
```
# To return the keys of a dictionary
for i in thisdict.keys():
    print(i)
```

```
brand
model
year
```

In [23]:
```
# To return values of a dictionary
for i in thisdict.values():
    print(i)
```

```
Ford
Mustang
1964
```

In [24]:
```
# Loop through both keys and values
for i in thisdict.items():
    print(i)
```

```
('brand', 'Ford')
('model', 'Mustang')
('year', 1964)
```

In [25]:
```
# Loop through both keys and values
for x, y in thisdict.items():
    print(x,y)
```

```
brand Ford
model Mustang
year 1964
```

# Add items to dictionary

In [26]:
```
faculty = {'Ram': 'C',
           'Shyam': 'Java',
           'Sita': 'Python',
           }
faculty
```

Out[26]:
```
{'Ram': 'C', 'Shyam': 'Java', 'Sita': 'Python'}
```

In [27]:
```
faculty[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-27-98ab19504337> in <module>
----> 1 faculty[0]

KeyError: 0
```

In [28]:
```
faculty['Sita']
```

Out[28]:
```
'Python'
```

In [29]:
```
# Using a new index key and assigning a value to it
faculty['Arjun'] = 'C++'
faculty
```

Out[29]:
```
{'Ram': 'C', 'Shyam': 'Java', 'Sita': 'Python', 'Arjun': 'C++'}
```

In [30]:
```
# Using update() method: The update() method will update the dictionary with the items from a gi
# If the item does not exist, the item will be added.
# The argument must be a dictionary, or an iterable object with key:value pairs.

faculty.update({"Rohan": "HTML"})
faculty
```

Out[30]:
```
{'Ram': 'C',
 'Shyam': 'Java',
 'Sita': 'Python',
 'Arjun': 'C++',
 'Rohan': 'HTML'}
```

In [31]:
```
faculty.update({"Rohan": "HTML", 'Raj': 'CSS', 'Rahul': 'Django'})
faculty
```

Out[31]:
```
{'Ram': 'C',
 'Shyam': 'Java',
 'Sita': 'Python',
 'Arjun': 'C++',
 'Rohan': 'HTML',
 'Raj': 'CSS',
 'Rahul': 'Django'}
```

# Change items in dictionary

In [32]:
```
faculty = {'Ram': 'C',
           'Shyam': 'Java',
           'Sita': 'Python',
           'Rohan': 'HTML',
           'Raj': 'CSS',
           'Rahul': 'Django'}
faculty
```

Out[32]:
```
{'Ram': 'C',
 'Shyam': 'Java',
 'Sita': 'Python',
 'Rohan': 'HTML',
 'Raj': 'CSS',
 'Rahul': 'Django'}
```

In [33]:
```
# using existing key
faculty['Sita'] = 'Ruby'
faculty
```

Out[33]:
```
{'Ram': 'C',
 'Shyam': 'Java',
 'Sita': 'Ruby',
 'Rohan': 'HTML',
 'Raj': 'CSS',
 'Rahul': 'Django'}
```

In [34]:
```
# using update() method
faculty.update({"Shyam": "HTML"})
faculty
```

Out[34]:
```
{'Ram': 'C',
 'Shyam': 'HTML',
 'Sita': 'Ruby',
 'Rohan': 'HTML',
 'Raj': 'CSS',
 'Rahul': 'Django'}
```

In [35]:
```
faculty.update({"Rohan": "Flask", 'Raj': 'SQL'})
faculty
```

Out[35]:
```
{'Ram': 'C',
 'Shyam': 'HTML',
 'Sita': 'Ruby',
 'Rohan': 'Flask',
 'Raj': 'SQL',
 'Rahul': 'Django'}
```

# Remove items in a dictionary

In [36]:
```
faculty = {'Ram': 'C',
           'Shyam': 'Java',
           'Sita': 'Python',
           'Rohan': 'HTML',
           'Raj': 'CSS',
           'Rahul': 'Django'}
faculty
```

Out[36]:
```
{'Ram': 'C',
 'Shyam': 'Java',
 'Sita': 'Python',
 'Rohan': 'HTML',
 'Raj': 'CSS',
 'Rahul': 'Django'}
```

In [37]:
```
# using del keyword: delete a key-value pair
del faculty['Shyam']
faculty
```

Out[37]:
```
{'Ram': 'C',
 'Sita': 'Python',
 'Rohan': 'HTML',
 'Raj': 'CSS',
 'Rahul': 'Django'}
```

In [38]:
```
# using pop() method: removes the item with the specified key name
faculty.pop('Raj')
faculty
```

Out[38]:
```
{'Ram': 'C', 'Sita': 'Python', 'Rohan': 'HTML', 'Rahul': 'Django'}
```

In [39]:
```
# using popitem() method: removes the last inserted item
faculty.popitem()
faculty
```

Out[39]:
```
{'Ram': 'C', 'Sita': 'Python', 'Rohan': 'HTML'}
```

In [40]:
```
# using clear() method: empties the dictionary
faculty.clear()
faculty
```

Out[40]:
```
{}
```

# Copy a dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

In [41]:
```
# Using copy() method
thisdict = {
              "brand": "Ford",
              "model": "Mustang",
              "year": 1964
            }
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

In [42]:
```
# Using dict() method
thisdict = {
               "brand": "Ford",
               "model": "Mustang",
               "year": 1964
            }
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

A dictionary key must be of a type that is immutable. For example, you can use an integer, float, string, or Boolean as a dictionary key. However, neither a list nor another dictionary can serve as a dictionary key, because lists and dictionaries are mutable. Values, on the other hand, can be any type and can be used more than once.

In [43]:
```
profile = {}
profile['fname']= 'Ram'
profile['lname']= 'Sharma'
profile['past_jobs'] = ['Accountant', 'SE', 'Analyst']
profile
```

Out[43]:
```
{'fname': 'Ram',
 'lname': 'Sharma',
 'past_jobs': ['Accountant', 'SE', 'Analyst']}
```

In [44]:
```
profile['past_jobs']
```

Out[44]:
```
['Accountant', 'SE', 'Analyst']
```

In [45]:
```
profile['past_jobs'][-1]
```

Out[45]:
```
'Analyst'
```

In [46]:
```
1 profile[['A', 'B']] = 10
2 profile
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-46-d6f2efcf1dbc> in <module>
----> 1 profile[['A', 'B']] = 10
      2 profile

TypeError: unhashable type: 'list'
```

In [47]:
```
1 profile[('A', 'B')] = 10
2 profile
```

Out[47]:
```
{'fname': 'Ram',
 'lname': 'Sharma',
 'past_jobs': ['Accountant', 'SE', 'Analyst'],
 ('A', 'B'): 10}
```

# Sorting a dictionary

In [48]:
```
1 # Displaying keys alphabeticallly
2
3 d = {'Ram': 45, 'Shyam': 12, 'Arjun': 45}
4 d_new = sorted(d.keys())
5 print(d_new, type(d_new))
```

```
['Arjun', 'Ram', 'Shyam'] <class 'list'>
```

In [49]:
```
# Sorting the Keys and Values in Alphabetical Order using the Key.

d = {'Ram': 45, 'Shyam': 12, 'Arjun': 45}
d_new = sorted(d.values())
print(d_new, type(d_new))
```

```
[12, 45, 45] <class 'list'>
```

In [50]:
```
d = {'Ram': 45, 'Shyam': 12, 'Arjun': 45}
d_new = sorted(d.items())
print(d_new, type(d_new))
```

```
[('Arjun', 45), ('Ram', 45), ('Shyam', 12)] <class 'list'>
```

In [51]:
```
# Creates a sorted dictionary (sorted by key)

from collections import OrderedDict

d = {'Ram': 45, 'Shyam': 12, 'Arjun': 45}
d_new = OrderedDict(sorted(d.items()))
print(d_new, type(d_new))

# Will see this further in collections module
```

```
OrderedDict([('Arjun', 45), ('Ram', 45), ('Shyam', 12)]) <class 'collections.OrderedDict'>
```

# Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries. Retrieving the values in the sublist or subdictionary requires an additional index or key.

In [52]:
```
myfamily = {
               "child1" :
                         {
                             "name" : "Emil",
                             "year" : 2004
                         },
               "child2" :
                         {
                             "name" : "Tobias",
                             "year" : 2007
                         },
               "child3" :
                         {
                             "name" : "Linus",
                             "year" : 2011
                         }
           }
myfamily
```

Out[52]:
```
{'child1': {'name': 'Emil', 'year': 2004},
 'child2': {'name': 'Tobias', 'year': 2007},
 'child3': {'name': 'Linus', 'year': 2011}}
```

In [53]:
```
child1 = {
  "name" : "Emil",
  "year" : 2004
}

child2 = {
  "name" : "Tobias",
  "year" : 2007
}

child3 = {
  "name" : "Linus",
  "year" : 2011
}


myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

myfamily
```

Out[53]:
```
{'child1': {'name': 'Emil', 'year': 2004},
 'child2': {'name': 'Tobias', 'year': 2007},
 'child3': {'name': 'Linus', 'year': 2011}}
```

In [54]:
```
myfamily['child1']
```

Out[54]:
```
{'name': 'Emil', 'year': 2004}
```

In [55]:
```
myfamily['child1']['name']
```

Out[55]:
```
'Emil'
```

In [56]:
```
myfamily['child2']['year']
```

Out[56]:
```
2007
```

In [57]:
```
1 myfamily.keys()
```

Out[57]:
```
dict_keys(['child1', 'child2', 'child3'])
```

In [58]:
```
1 myfamily.values()
```

Out[58]:
```
dict_values([{'name': 'Emil', 'year': 2004}, {'name': 'Tobias', 'year': 2007}, {'name': 'Linus', 'y
ear': 2011}])
```

In [59]:
```
1 myfamily.items()
```

Out[59]:
```
dict_items([('child1', {'name': 'Emil', 'year': 2004}), ('child2', {'name': 'Tobias', 'year': 200
7}), ('child3', {'name': 'Linus', 'year': 2011})])
```

In [60]:
```
1 for i in myfamily:
2     print(i, myfamily[i])
```

```
child1 {'name': 'Emil', 'year': 2004}
child2 {'name': 'Tobias', 'year': 2007}
child3 {'name': 'Linus', 'year': 2011}
```

In [61]:
```
1 for i in myfamily.values():
2     print(i,type(i))
```

```
{'name': 'Emil', 'year': 2004} <class 'dict'>
{'name': 'Tobias', 'year': 2007} <class 'dict'>
{'name': 'Linus', 'year': 2011} <class 'dict'>
```

In [62]:
```
1 for i in myfamily.values():
2     print(i.keys())
```

```
dict_keys(['name', 'year'])
dict_keys(['name', 'year'])
dict_keys(['name', 'year'])
```

In [63]:
```
for i in myfamily.values():
    print(i.values())
```

```
dict_values(['Emil', 2004])
dict_values(['Tobias', 2007])
dict_values(['Linus', 2011])
```

In [64]:
```
for i in myfamily.values():
#     print(i)
    for j in i:
        print(j, i[j])
```

```
name Emil
year 2004
name Tobias
year 2007
name Linus
year 2011
```

# Dictionary Methods

clear() : Removes all the elements from the dictionary
copy() : Returns a copy of the dictionary
fromkeys() : Returns a dictionary with the specified keys and value
get() : Returns the value of the specified key
items() : Returns a list containing a tuple for each key value pair
keys() : Returns a list containing the dictionary's keys
pop() : Removes the element with the specified key
popitem() : Removes the last inserted key-value pair
setdefault() : Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update() : Updates the dictionary with the specified key-value pairs
values() : Returns a list of all the values in the dictionary

In [65]:
```
# fromkeys(keys,value): value is optional argument, default value is None

x = ('key1', 'key2', 'key3')
y = 0
thisdict = dict.fromkeys(x, y)

print(thisdict)
```

```
{'key1': 0, 'key2': 0, 'key3': 0}
```

In [66]:
```
# fromkeys(keys,value)

x = ('key1', 'key2', 'key3')
thisdict = dict.fromkeys(x)
print(thisdict)
```

```
{'key1': None, 'key2': None, 'key3': None}
```

In [67]:
```
# setdefault(keyname, value): value is optional argument, default value is None

car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("model", "Bronco")
print(x)
print(car)
```

```
Mustang
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

In [68]:
```
# setdefault(keyname, value): value is optional argument, default value is None

car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("color", "white")
print(x)
print(car)
```

```
white
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'white'}
```

In [ ]:
```

```