Lecture 4.2 : Dictionaries 2

Sorting dictionary items on keys

- Suppose we want to print the contents of a dictionary sorted on keys. For example, suppose we want to print out the contents of our phone book in increasing alphabetical order of the names (keys). How would we do that? Well let's try calling `sorted()` on the dictionary's item list and see if that puts things in the desired order.

```
phone_book = { 'joe' : '086 7346659',
   'jimmy' : '085 2313872',
   'cindy' : '087 9452238' }

print(phone_book)
print(phone_book.items())
print(sorted(phone_book.items()))
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
dict_items([('joe', '086 7346659'), ('jimmy', '085 2313872'), ('cindy', '087 9452238')])
[('cindy', '087 9452238'), ('jimmy', '085 2313872'), ('joe', '086 7346659')]
```

- Great. Calling `sorted()` on the items in the phone book sorts them in increasing order of the first member of each returned tuple i.e. the name. This is exactly what we want. So let's employ that approach.

```
for k, v in sorted(phone_book.items()):
   print(f'{k} ---> {v}')
```

```
cindy ---> 087 9452238
jimmy ---> 085 2313872
joe ---> 086 7346659
```

Sorting dictionary items on values

- Suppose we want to print the contents of a dictionary sorted not on keys but on values. How can we do that? Just calling `sorted()` on items won't work as it sorts on the name.

```
print(phone_book.items())
print(sorted(phone_book.items()))
```

```
dict_items([('joe', '086 7346659'), ('jimmy', '085 2313872'), ('cindy', '087 9452238')])
[('cindy', '087 9452238'), ('jimmy', '085 2313872'), ('joe', '086 7346659')]
```

- So what can we do? We need to tell `sorted()` to sort on the *second* component of each item returned by `items()` (the phone number). Can we do that?
- We can sort on an arbitrary data member of an object by specifying a custom `key()` function when we invoke the `sorted()` function.
- It is the job of this `key()` function to return the item we wish to sort on.
- In the above example we wish to sort on the second item in a tuple so our `key()` function should be defined as shown below.

```
def tagger(item):
   return item[1]

# sort on the value (i.e. phone number)
for k, v in sorted(phone_book.items(), key=tagger):
   print(f'{k} ---> {v}')
```

```
jimmy ---> 085 2313872
joe ---> 086 7346659
cindy ---> 087 9452238
```

Key function intuition

- Imagine we wanted to sort all students in a class on how far they lived from DCU. We would ask each student “How far do you live from DCU?”. We would tag each student with their answer e.g. 5 if they lived 5 miles from DCU, 4 if they lived 4 miles away, etc.
- With an appropriate tag attached to each student, we would then sort them in increasing/decreasing order of their tags.
- Passing a `key()` function to `sorted()` does a similar tagging job. The job of the `key()` function is to tag each object passed to it. For each object passed to it, it returns the corresponding tag.
- Above we are passing a tuple to the `key()` function (consisting of a dictionary key and value). The `key()` function says “sort on the value” by returning the value.

More sorting examples

- Below we create a new dictionary and print its contents sorting on keys and values in ascending and descending order.

```
zoo = { 'snakes' : 20,
 'hippos' : 2,
 'tarantulas' : 15,
 'zebras' : 7 }

def tagger(item):
   return item[0]

# increasing key order
for k, v in sorted(zoo.items(), key=tagger):
   print(f'{k} ---> {v}')
```

```
hippos ---> 2
snakes ---> 20
tarantulas ---> 15
zebras ---> 7
```

```
# decreasing key order
for k, v in sorted(zoo.items(), key=tagger, reverse=True):
    print(f'{k} ---> {v}')
```

```
zebras ---> 7
tarantulas ---> 15
snakes ---> 20
hippos ---> 2
```

```
def tagger(item):
    return item[1]

# increasing value order
for k, v in sorted(zoo.items(), key=tagger):
    print(f'{k} ---> {v}')
```

```
hippos ---> 2
zebras ---> 7
tarantulas ---> 15
snakes ---> 20
```

```
# decreasing value order
for k, v in sorted(zoo.items(), key=tagger, reverse=True):
    print(f'{k} ---> {v}')
```

```
snakes ---> 20
tarantulas ---> 15
zebras ---> 7
hippos ---> 2
```

Tabulating dictionary keys and values

- Suppose we want to both sort and neatly print dictionary keys and corresponding values. How can we do that?
- Firstly we need to work out the width of the widest value in our dictionary. We can do so as shown below.

```
print(zoo.values())
print(max(zoo.values()))
print(str(max(zoo.values())))
print(len(str(max(zoo.values()))))
max_v_width = len(str(max(zoo.values())))
```

```
dict_values([20, 2, 15, 7])
20
20
2
```

- Next we need to work out the width of the widest key in our dictionary. We can do so as shown below (note again the use of an appropriate `key()` function).

```
print(zoo.keys())
print(max(zoo.keys()))
print(max(zoo.keys(), key=len))
print(len((max(zoo.keys(), key=len))))
max_k_width = len((max(zoo.keys(), key=len)))
```

```
dict_keys(['snakes', 'hippos', 'tarantulas', 'zebras'])
zebras
tarantulas
10
```

- We now format our output using the above widths.

```
for k, v in sorted(zoo.items()):
   print(f'{k:>{max_k_width}s} ---> {v:>{max_v_width}d}')
```

```
    hippos --->  2
    snakes ---> 20
tarantulas ---> 15
    zebras --->  7
```

Other dictionary methods

- There are more dictionary methods. You can look them up using `help` or the `pydoc` command.

```
help(dict)
```

```
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
```

```
|      in the keyword argument list.  For example:  dict(one=1, two=2)
|
|  Built-in subclasses:
|      StgDict
|
|  Methods defined here:
|
|  __contains__(self, key, /)
|      True if the dictionary has the specified key, else False.
|
|  __delitem__(self, key, /)
|      Delete self[key].
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(...)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the dict keys.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(...)
|      D.__sizeof__() -> size of D in memory, in bytes
|
|  clear(...)
|      D.clear() -> None.  Remove all items from D.
|
|  copy(...)
|      D.copy() -> a shallow copy of D
|
|  get(self, key, default=None, /)
|      Return the value for key if key is in the dictionary, else default.
|
```

```
|  items(...)
|      D.items() -> a set-like object providing a view on D's items
|
|  keys(...)
|      D.keys() -> a set-like object providing a view on D's keys
|
|  pop(...)
|      D.pop(k[,d]) -> v, remove specified key and return the corresponding value.
|      If key is not found, d is returned if given, otherwise KeyError is raised
|
|  popitem(self, /)
|      Remove and return a (key, value) pair as a 2-tuple.
|
|      Pairs are returned in LIFO (last-in, first-out) order.
|      Raises KeyError if the dict is empty.
|
|  setdefault(self, key, default=None, /)
|      Insert key with a value of default if key is not in the dictionary.
|
|      Return the value for key if key is in the dictionary, else default.
|
|  update(...)
|      D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
|      If E is present and has a .keys() method, then does:  for k in E: D[k] = E[k]
|      If E is present and lacks a .keys() method, then does:  for k, v in E: D[k] =
|      In either case, this is followed by: for k in F:  D[k] = F[k]
|
|  values(...)
|      D.values() -> an object providing a view on D's values
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  fromkeys(iterable, value=None, /) from builtins.type
|      Create a new dictionary with keys from iterable and values set to value.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```