19-dict-I

November 8, 2020

1 Dictionaries

Dictionaries are a very useful kind of data-structure. They work more or less like lists, except that instead of indexing elements by a number (like `L[4]`), you can index them using almost anything (strings, for example: `D["Qatar"]`). More precisely, a dictionary associates *keys* (the indices) with *values*. Keys are unique and are associated with only one value.

Another way of seeing a dictionary is as a set of key/value pairs. Note that there is no implicit order on the pairs, and the *keys are unique*.

1.1 Creating dictionaries

A concrete dictionary can be declared using curly braces and colons, like this:

[1]:
```
# COVID-19 infected persons, as of Nov. 04, 2020
covid = {"Qatar": 133370,
         "USA": 9706838,
         "Italy": 790377,
         "Iran": 646164,
         "South Korea": 26925,
         "Oman": 116847,
         "Egypt": 108122,
         "Jordan": 91234,
         "Lebanon": 87097,
         "India": 8353493
        }
```

To the left of evey colon is the "key" and to the right of it is the "value".

Dictionaries can also be created from a list of pairs.

[2]:
```
L = [("Blue", 10), ("Red", 7), ("Green", 15), ("Black", 12)]
colors = dict(L)
print(colors)
```

```
{'Blue': 10, 'Red': 7, 'Green': 15, 'Black': 12}
```

The **empty** dictionary is created either using the function `dict()` or empty curly braces `{}`.

[3]:
```
d1 = dict()
d2 = {}
print(d1)
print(d2)
```

```
{}
{}
```

1.2 Adding entries

Once you have a dictionary (possibly an empty one), you can add entries to it by assigning values using appropriate indexes:

[4]:
```
capitals = dict()
capitals["Qatar"] = "Doha"
capitals["Germany"] = "Berlin"
capitals["Vietnam"] = "Hanoi"
capitals["Nigeria"] = "Lagos"
print(capitals)
```

```
{'Qatar': 'Doha', 'Germany': 'Berlin', 'Vietnam': 'Hanoi', 'Nigeria': 'Lagos'}
```

Observe that you can add an entry by defining a non-existent key (i.e., a key that does not exist yet in the dictionary), index the dictionary through that key, and assign a value (any valid value with a valid type) to it. If you assign a value to an existing key, the old value is **overwritten**. Hence, dictionaries are **mutable** (similar to lists but dissimilar to tuples and strings).

[5]:
```
capitals["Nigeria"] = "Abuja"
print(capitals)
```

```
{'Qatar': 'Doha', 'Germany': 'Berlin', 'Vietnam': 'Hanoi', 'Nigeria': 'Abuja'}
```

1.3 Getting values

Values can be obtained by indexing the dictionary through keys:

[6]:
```
v = capitals["Nigeria"]
print(v)
```

```
Abuja
```

If the key is not there, python raises a `KeyError`.

[7]:
```
v = capitals["Russia"]
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)
```

```
    <ipython-input-7-9951b880e302> in <module>()
----> 1 v = capitals["Russia"]


    KeyError: 'Russia'
```

To avoid `KeyError` errors, you can check if the key is in the dictionary before trying to access its corresponding value (see below), or you can use the `get()` function. This function is useful if you want to get a default value in case the key is non-existent. The first parameter of 'get()' is the key of the value that you want to access and the second parameter is the default value that will be returned in case the key is non-existent.

[8]:
```
v1 = covid.get("Qatar", 0)
print("Number of cases in Qatar:", v1)

v2 = covid.get("Easter Island", 0)
print("Number of cases in Easter Island:", v2)
```

```
Number of cases in Qatar: 470
Number of cases in Easter Island: 0
```

1.3.1 d.values()

We can get all the values in a dictionary via calling the `values()` function, which returns an "object" containing all the values in the dictionary.

[9]:
```
vals = capitals.values()
print(vals)
```

```
dict_values(['Doha', 'Berlin', 'Hanoi', 'Abuja'])
```

Observe that there is a list inside the returned object. You can get this list by simply wrap it with `list()`:

[10]:
```
lvals = list(vals)
print(lvals)
```

```
['Doha', 'Berlin', 'Hanoi', 'Abuja']
```

1.4 Getting keys

There is no direct way to get one particular key from the dictionary (since they are not indexed like values).

To check if a *key* is in a dictionary, you can use `in`/`not in`. This is vey useful to avoid the `KeyError` shown above!

[11]:
```
"Qatar" in capitals
```

[11]: True

[12]:
```
"Jordan" not in capitals
```

[12]: True

1.4.1 d.keys()

We can get all keys in a dictionary via calling the function `keys()`. Similar to `values()`, it returns an object with all the keys in the dictionary. Again, to get a list out of it, wrap it with `list()`.

[13]:
```
keys = list(capitals.keys())
print(keys)
```

```
['Qatar', 'Germany', 'Vietnam', 'Nigeria']
```

1.5 Number of entries

The function `len(d)` returns the number of entries in the dictionary.

[14]:
```
print(len(capitals))
```

```
4
```

1.6 Removing entries

An entry at key `k` can be removed from dictionary `d` via the `del` command:

[15]:
```
del covid["Oman"]
print(covid)
```

```
{'Qatar': 470, 'US': 19624, 'Italy': 47021, 'China': 81286, 'Iran': 19644,
'South Korea': 8652, 'Egypt': 285, 'Jordan': 85, 'Lebanon': 177, 'Philipines':
230, 'India': 250}
```

1.7 Looping through dictionaries

We can loop through dictionaries using a `for` loop, where the loop variable will range among the dictionary's keys.

[16]:
```
for country in capitals:
        print("country =", country)
        if capitals[country] == "Hanoi":
            print(country + "'s capital is Hanoi")
```

```
country = Qatar
country = Germany
country = Vietnam
```

```
Vietnam's capital is Hanoi
country = Nigeria
```

1.8 Exercise 1

Ali recently got a 3D printer, and decided to open a business for printing messages in 3D letters. Printing with those printers is kind of a slow, so he would like to group letter for printing. For example, if the message is `"Congratulations, Ahmad!"`, Ali would like to print 3 `"a"`s at once.

Help Ali figure out how much of each character he needs to print. Implement the function `charFreq(s)` that takes a string (the message) as a parameter, and returns a dictionary where the keys are characters, and values are the number of times Ali needs to print them. Remember that: - spaces do not need to be printed - capitalization matters (i.e. `"a"` is different from `"A"`) - punctuation needs to be printed

[17]:
```
def charFreq(s):
    return {}
```

1.9 Exercise 2

Suppose you have a dictionary `d` of COVID-19 cases as the one above, where the keys are countries and the values are the numbers of cases in these countries. Implement the function `sortByCases(d)` that returns a list of countries in decreasing order of COVID-19 cases.

[18]:
```
def sortByCases(d):
    return []
```