Python: Dictionaries

1405 Instructor: Ruiqing (Sam) Cao

Collection Types: Dictionary (`dict`)

- A `dictionary` stores data as an **unordered, mutable** collection of *key-value pairs*

➢Think: a finite space hotel with rooms referenced by ***unique*** names like “Torsten” or “A750” → *That’s what a dictionary is !*

E.g., `{'ID':10,'name':'Alice','isFemale':True}` is a dictionary

→ A `dict` type enclosed by braces **{}** and key-value pairs linked by colon **:**

Collection Types: Dictionary (`dict`)

Properties of keys and values

- *Keys* can be strings, numbers, or tuples (but not lists)
- *Keys* must be unique
- *Values* can be of any data type

Dictionaries are *unordered*

- The *keys are not in any specific order*, and values are called by referencing the key that points to that value

Create a Dictionary

- Creating an *empty dictionary*: `di={}` or `di=dict()`

- Creating a *dictionary with items*: `di={'a':0,'b':1}`

Access & Update a Dictionary

First, create a dictionary and store it: `di={'a':0,'b':1}`

- *Access* a value in a dictionary by referencing its key (*must exist*)

`di['a']` → `0` `di['b']` → `1`

- Think: find people in a hotel room by referencing the room name

- *Update* a value referenced by an *existing* key

```
di['b']=-1
```

→ Don't use the term *"index"* which is for *ordered* sequence only; The equivalent of an "index" in a dictionary is a ***key***

Add an Item to a Dictionary

First, create a dictionary and store it: `di={'a':0,'b':1}`

- *Add* a new key-value pair to the dictionary simply by typing

```
di[new_key]=new_value
```

- If `new_key` *does not exist* among keys of the dictionary `di`, the statement above adds an item `new_key:new_value` to `di`
- If `new_key` *already exists* as a key, the statement above simply updates the value referenced by `new_key` into `new_value`

Get Keys, Values, Items, & Length

- Get *all keys* of a dictionary as a list

```
list(di.keys())
```

- Get *all values* of a dictionary as a list

```
list(di.values())
```

- Get *all items (key-value pairs)* of a dictionary as a list (*returns a list of key-value pairs as tuples*)

```
list(di.items())
```

- Get *number of items* (key-value pairs) in a dictionary

```
len(di)
```

Check for Dictionary Membership

- Check whether an object is a key of the dictionary

  `key in di` or `key in di.keys()`

- Check whether an object is a value of the dictionary

  `value in di.values()`

- Check whether a key-value pair is an item of the dictionary

  `(key, value) in di.items()`

➢ *Return Boolean type* **`True`** *or* **`False`**

Iterate Through a Dictionary

- Iterate through all keys of a dictionary

```
for key in di:
    ...
```

or

```
for key in di.keys():
    ...
```

- Iterate through all values of a dictionary

```
for value in di.values():
    ...
```

- Iterate through all key-value pairs of a dictionary

```
for key, value in di.items():
    ...
```

Exercises: Dictionaries

Write a Python program to:

1. Ask the user to input a paragraph.
2. Convert the paragraph to all lowercase.
3. Replace all non-alphabetical characters (not a-z) by whitespace [hint: `st.isalpha()`].
4. Split the paragraph into words.
5. Creates a dictionary that counts word frequencies.
6. Prints the contents of the dictionary as a list of tuples, in descending order of word frequencies.

Combine Dictionaries

- You can use the method `update()` to add all items in `di2` to `di1` (note: method *directly modifies* `di1` without returning an object)

```
di1.update(di2)
```

- *Return the combined dictionary as an object* without modifying the original dictionaries

```
bothdi = di1 | di2
```
(Requires Python version 3.9+)

```
bothdi = {**di1, **di2}
```
(Requires Python version 3.5+)

Use Case: Dictionaries & Tabular Data

- You can use a dictionary to store *one row of a tabular data set* with *variable names*

For instance, examine this dictionary

```
{'gvkey':33175,'name':'SPOTIFY','year':2019,'sale_bn':7.0}
```

It contains the same information as a row in the table below (note that dictionary `key` → *variable name* and `value` → *variable value*

gvkey (int)	name (str)	year (int)	sale_bn (float)
33175	SPOTIFY	2019	7.0

→ Very similar to the structure of a ***JSON string***, which are extremely widely used for storing semi-structured data (more on this later)

Review of Basics: Python Data Structures