PYTHON: TUPLES, DICTIONARIES AND SETS

Objectives

- To understand two Python data models – Tuple and Dictionary
- To apply these data models to solve various types of problems, such as counting the frequency of words

Tuples

- A **tuple** is an *immutable* linear data structure.
  - The elements in a tuple cannot be modified.
- You can create a tuple in Python by
  - `myList = (1, 2, 3, 4)`
  - `myGrades = ("A+", "A", "B+", "B")`
  - `myMenu = ("Sausage", "egg", "bread", "potato")`
  - `myMix = (1, "Spam ", 4, "U")`
  - `myEmptiness = ()`
  - `mySingleton = (1,)`
- Note: tuples of one element must include a comma following the element, e.g., `(1,)` (instead of simply `(1)`).
  - Why?

EXERCISE 7.1

Try:

- `x = (1, 2, 3, 4)` and then `x = (1, 2, 3, 4)` again and check whether the two tuples are stored in the same location using `id()`.
- `x[0] = 10`
- `y = (9)` and `z = (9,)` and use `type()` to find out their types.

EXERCISE 7.2

Try `append()`, `insert()`, `sort()`, `reverse()`, and `del()` on a tuple, e.g., `x` to see how these methods work for tuples (as compared with lists).

Storing students' records

- Problem: How to store student's data in a program after reading the data from a file?
- A possible way is to store the data in a list.
- However, a list can be indexed only by non-negative integers, but ours should be indexed by names.
- A common solution in most programming languages:
  - Map the names to a set of integers.
  - Use the integers as indices for accessing the list.
- A much better solution in Python:
  - Use a Python *dictionary* that maps the set of names (keys) directly to a set of values.

EXERCISE 7.3

Try

```
nicknames = dict()
print(nicknames)
nicknames = {"Mickey":"Mickey Mouse in Disney",
"Minnie":"Minnie Mouse in Disney"}
print(nicknames)
nicknames["Woody"] = "Woody in Toy Story"
print(nicknames)
```

Here, "Mickey" is the key and "Mickey Mouse in Disney" the value.

Python dictionary

- A dictionary is a mutable, associative data structure of variable length.
  - The key can be of any immutable type.
  - The values can be of any type and are unordered.
- Examples:
  - `dictA = {'Mickey': 'Mickey Mouse in Disney', 1:2, ('a', 'b', 'c'): [1, 2, 3]}`
  - The third element has a key of tuple key and a value of list.
  - `dictB = {'Mickey': 'Mickey Mouse in Disney', 1: 2, ('a', 'b'): [1, 2, 3, 4]}`
  - You can get the value using its index, e.g., `dictA["Mickey"]`, `dictB[('a', 'b')]`.

EXERCISE 7.4

Create a dictionary of dictionary, such as

```
dictC = {"a":1, "b":{"aa":11, "bb":22}}.
```

How do you get the values in the inner dictionary (i.e., 11 and 22)?

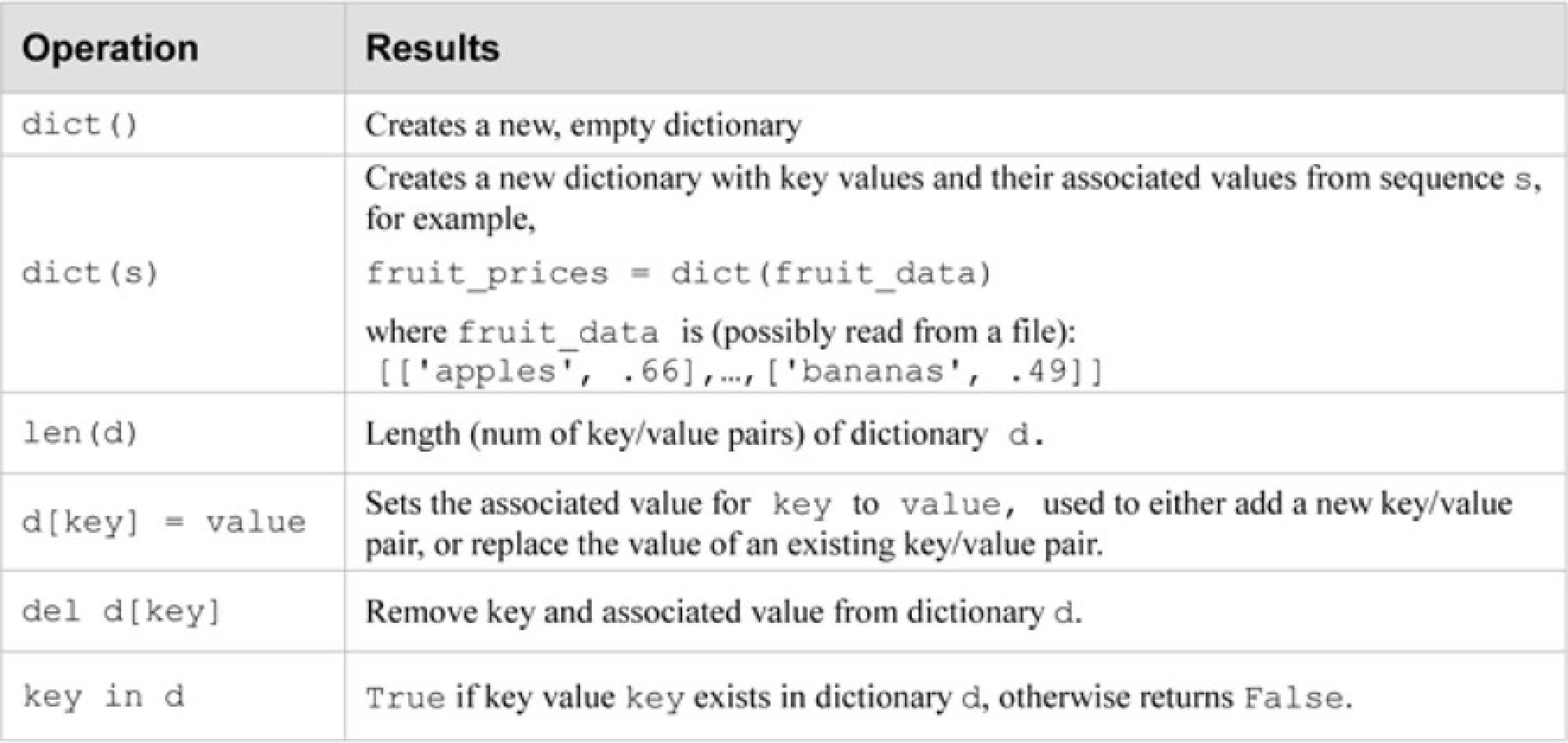
EXERCISE 7.5

Create a dictionary `dictD` by storing your own set of values. Then try

- `len(dictD)`
- Use the keyword "**`in`**" to check whether an object is a key in `dictD`.
- Use a `for` loop to print out all the keys in `dictD`.
- Use a `for` loop to print out all the keys and the corresponding values, one pair on a line.

(Hints on next slide)

Dictionary operators



Operation	Results
`dict()`	Creates a new, empty dictionary

`dict(s)`	Creates a new dictionary with key values and their associated values from sequence `s`, for example,
	`fruit_prices = dict(fruit_data)`
	where `fruit_data` is (possibly read from a file):
	`[['apples', .66],…,['bananas', .49]]`

`len(d)`	Length (num of key/value pairs) of dictionary `d`.
`d[key] = value`	Sets the associated value for `key` to `value`, used to either add a new key/value pair, or replace the value of an existing key/value pair.
`del d[key]`	Remove key and associated value from dictionary `d`.
`key in d`	`True` if key value `key` exists in dictionary `d`, otherwise returns `False`.

Source: Charles Dierbach. 2013. Introduction to Computer Science Using Python. Wiley.

Dictionary methods

- Python provides a number of other methods that iterate through the elements in a dictionary.
  - `items()`: Returns all the key-values pairs as a list of tuples
  - `keys()`: Returns all the keys as a list
  - `values()`: Returns all the values as a list
- The objects returned by `dict.keys()`, `dict.values()` and `dict.items()` on a dictionary `dict` are view objects.
  - They provide a dynamic view on the dictionary’s entries, which means that when the dictionary changes, the view reflects these changes.
  - They are iterable; therefore they can be used in a loop.

EXERCISE 7.6

Create a dictionary and use the three methods in the previous slide and a loop to print out their keys only, values only, and key-value pairs.

EXERCISE 7.7

Write a function **`count(str)`** that will count the number of occurrences of each word in a long string and store them in a dictionary. Print the key-value on a new line. Assume that the string does not contain any punctuations, symbols, and numbers, e.g., **`str`** = "**`a for apple b for boy c for cat d for dog`**" will yield a count of 4 for "**`for`**" and 1 for each other word.

EXERCISE 7.8

In this exercise, we would like to print out the statistics in a sorted order of the words. In order to do this, you need to create a list of keys first for the key-value in the dictionary and then apply `sorted()` on the list of keys before using the keys to print.

EXERCISE 7.9

Given a dictionary *d* and a key *k*, it is easy to find the corresponding value *v = d[k]*. This operation is called a lookup. You have been performing lookup in 7.7 and 7.8 already. Write a function **`reverse_lookup(d, v)`** for reverse lookup. That is, given *d* and a value *v*, the function will return the first key that maps to *v*, returning **`None`** or "not found" as needed.

Using our example string, reverse lookup for 4 will be "for". Reverse lookup for 1 *might* yield "dog". Reverse lookup for 3 will be `None`.

EXERCISE 7.10

Modify **`reverse_lookup(d, v)`** in exercise 7.9 so that it builds and returns a list of *all* keys that map to *v*, or an empty list if there are none.

END