

Handout 8

Dictionaries. Sets.

DICTIONARY

Dictionary, a.k.a hash map or associative array, is a flexibly-sized collection of **key-value pairs**, where key and value are Python objects.

Think of a dictionary as a quick look-up table, where you look-up information by its **key**. (There is no look-up operation from *value* to the key.)

Some examples: `dictDemo.py`

```
def main():
    stock_prices_dict = {      # Dictionary with string keys and numeric values
        "AAPL": 135.25,
        "GOOGL": 2500.75,
        "MSFT": 240.50,
        "AMZN": 3250.00,
        "FB": 310.75
    }

    # Each month is associated with a list of sales data (numbers)
    sales_data_dict = {        # for different products or categories.
        "Jan": [10000, 15000, 12000, 18000],
        "Feb": [11000, 16000, 13000, 19000],
        "Mar": [12000, 17000, 14000, 20000]
    }

    sales_by_rep_data_dict = {
        "Jan": {"Joe": 10000, "Jen": 15000, "Emma": 18000},
        "Feb": {"Emma": 1300, "Greg": 15000},
        "Mar": {"Joe": 10300, "Greg": 15000, "Jeff": 18000}
    }

    person_info = {           # Dictionary with mixed data types
        "name": "John Doe",
        "age": 35,
        "is_employed": True,
        "height": 175.5,
        "languages_spoken": ["English", "Spanish"],
        "contact_details": {
            "email": "john@example.com",
            "phone": "+1234567890",
            "address": {
                "street": "123 Main St",
                "city": "Anytown",
                "zipcode": "12345"
            }
        }
    }

    code_dict = {
        101: "apple",
        340: "orange",
        450: "grape",
        25: "banana",
        6000: "watermelon"
    }
```

Important rules:

- Access is by key, e.g.
`google_price = stock_prices_dict["GOOGL"]`
- Keys are distinct (i.e. no duplicate keys)
- Key-value pairs are not stored in any specific order.
- Keys must be of an immutable type (e.g. int, string, tuple).
- Value can be of *any* type.
- Create a dictionary by using curly braces {} and listing key:value pairs separated by a comma:

```
dict = {} # Create an empty dictionary
dict = {"john":40, "peter":45} # Create a dictionary
```

- To **add** an entry to a dictionary: `dictionary[key] = value`, e.g.
`dict["susan"] = 50`
- To **delete** an entry from a dictionary, use `del dictionary[key]`
- To check if a key is in the dictionary: **in**

PRACTICE PROBLEMS

1. Experiment with adding and modifying values in the dictionaries in **dictDemo.py**
2. Define a function with no parameters, which reads people's names and an associated number (e.g. distance traveled to work), until user enters an empty string and creates and returns a dictionary of entries *name :distance*.
3. Assume the same email can be entered more than once – change the program
 - a) to keep only the first number entered with the repeated email,
 - b) to keep all numbers

Method(params):returns	Description
d.keys():dict_keys	Returns a sequence of keys.
d.values():dict_values	Returns a sequence of values.
d.items():dict_items	Returns a sequence of tuples (key, value).
d.clear(): None	Deletes all entries.
d[key]	Returns the value for the key.
d.get(key): value	
d.pop(key): value	Removes the entry for the key and returns its value.

Examples:

- **how to iterate over dictionaries**

<code>for key in dict.keys():</code>	Iterate through the keys of the dictionary
<code>for key in sorted(dict.keys()):</code>	Iterate through the keys of the dictionary in sorted order
<code>for value in dict.values():</code>	Iterate through the values of the dictionary

```
for key, value in dict.items():
for itemTuple in dict.items():
```

Iterate through the key/value pairs of the dictionary

- often helps to turn the keys or values into lists, e.g. `lst = list(dict.keys())`

PRACTICE PROBLEMS

4. Create a function within `dictDemo.py` to do each of the following
 - a. Compute the max stock price and print out the name of stock(s) with that price
 - b. Compute and print the total sales from `sales_data_dict`
 - c. Add a record in `sales_data_dict` for April, with value [100, 200]
 - d. Add a record in `sales_by_rep_data_dict` for March reflecting sales by Amy of \$3000
 - e. Print out monthly sales totals based on `sales_by_rep_data_dict`
 - f. Create a sorted list of values in the `code_dict`.
5. Add code to program developed in problem 2 to find the names of the people who travels more than an average distance (assuming there is one such person).
6. Implement with a dictionary: File `weeklyTasks.txt` has day names followed by task descriptions:

```
Mon -- office hours
Tue - meeting
Mon - research meeting
Wed - classes
Thu -- office hours
Fri - meetings
Thu - faculty meeting
Fri - paper deadline
```

Create a program which will

- A. read the `weeklyTasks.txt` file, collecting all tasks for a day under the same dictionary entry, so, for example, the value for `Mon` would be a list `["office hours", "research meeting"]`
This functionality should be implemented in a function `createDictFromFile()`, which must be passed a path to a file as a parameter, and returns the dictionary.
- B. Ask the user to specify the day, then retrieve and print the day's task according to the constructed dictionary.

This functionality should be implemented in a function `displayTask()`, which must be passed the dictionary associating days with tasks, as a parameter, and runs the loop displaying the tasks for the user's specified day.

Include a main function that starts the program and calls the two functions in A,B, appropriately.

SET

A set is an *unordered* collection with *no duplicate elements*. Sets are mutable, supporting add/remove/Basic uses include membership testing and eliminating duplicate entries.

Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Create sets using curly braces {} or the set() function .

Note: to create an **empty set you have to use set()**, not {}; the latter creates an empty **dictionary**.

To eliminate duplicates from any collection **c** – create a set out of it, calling set(**c**)

Examples:

```
>>> lst = [1,2,3,3,3,2,1]
>>> set(lst)
{1, 2, 3}

>>> words = "To be or not to be".lower().split()
['to', 'be', 'or', 'not', 'to', 'be']
>>> distinctWords = set(words)
{'be', 'not', 'or', 'to'}
>>> sorted(distinctWords)
['be', 'not', 'or', 'to']
```

Operations on sets: in, not in, len(), max(), min()

Set operations as methods: intersection, union, difference

Set operations as operations: &, |, -

method	operation	description
a.union(b)	a b	Set of items in a or b
a.intersection(b)	a & b	Set of items in both a and b
a.difference(b)	a - b	Set of items in a that are not in b

Set-modifying methods:

```
s.add (e):          add element e to set s.
s.remove (e):       remove element e from set s; error if e is not in s.
s.discard (e):      remove element e from set s, if it is present.
s.clear()           removes all elements from s
```

Examples:

```
>>> set1 = {'green', 'blue', 'red'}
>>> set2 = {'green', 'blue', 'yellow'}
>>> set1.union(set2)
{'blue', 'red', 'yellow', 'green'}
>>> set1 | set2
{'blue', 'red', 'yellow', 'green'}

>>> set1.intersection(set2)
{'blue', 'green'}
>>> set1 & set2
{'blue', 'green'}
```