

Information, Computation, Communication

Learning Python

Dictionaries

Agenda

- [What is a dictionary](#)
- [Initializing, reading, or writing](#)
- Examples of dictionary usage
 - [Grouping and counting items](#)
 - [Mapping a key to multiple values](#)
- Creating a dictionary
 - [fromkeys\(\)](#) and [dict\(\)](#)
- [Extracting keys or values to a list](#)
- [Looping using items\(\)](#)
- [Sorting](#)
- [Removing items](#)

Dictionaries vs. Lists

Dictionaries and lists are similar

- Both are **mutable**
- Both are **dynamic** (can grow and shrink as needed)
- Both can be **nested** (a list can contain another list or dictionary)

What are the differences, then?

- The main difference lies in how the elements are accessed
 - **List** elements are accessed via their **index** in the list
 - **Dictionary** elements are key-value pairs
 - To access a value, a **key** is needed

Dictionaries

- Dictionaries are, so called, **associative** arrays
- Collections of **key:value** pairs
- **Like lists:** Commas separate the elements (i.e., key-value pairs)
- **Unlike lists, like sets:** Dictionaries are delimited by curly braces

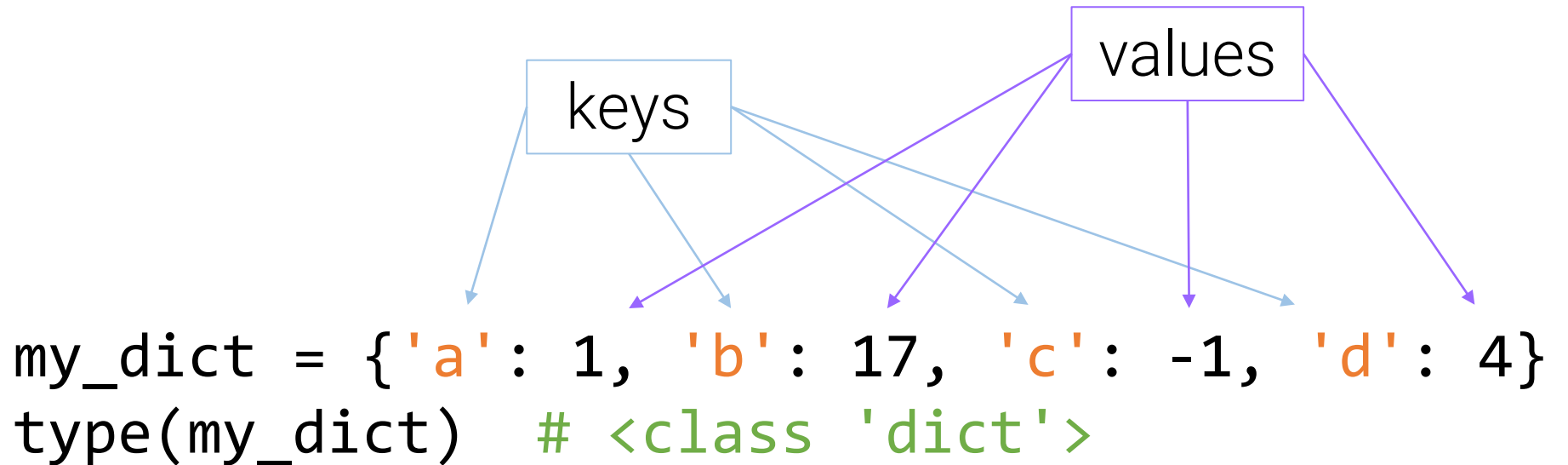
keys	values
'a'	1
'b'	17
'c'	-1
'd'	4

```
my_dict = {'a': 1, 'b': 17, 'c': -1, 'd': 4}
type(my_dict) # <class 'dict'>
```

Dictionaries

- Dictionaries are, so called, **associative** arrays
- Collections of **key:value** pairs
- Each key-value pair maps the key to its associated value

keys	values
'a'	1
'b'	17
'c'	-1
'd'	4



Dictionaries

- The values in dictionary elements can be of **any data type**
- Dictionaries are **unordered**
 - There is no built-in notion of order (i.e., indices) as values are accessed through their keys and not their location in the dictionary
- Any **immutable** Python object type can be used as a **key**
 - Key should not be modifiable (mutable)
 - For example, a list cannot be a key because a list can be modified

Creating an empty dictionary

```
my_dict = {}
```

Recall: creating an empty set, which also uses braces

```
my_set = set()
```

Initializing/Reading From a Dictionary

Initializing with key-value pairs

```
my_dict = {'a': 1,  
           'b': 17,  
           'c': -1,  
           'd': 4}
```

keys	values
'a'	1
'b'	17
'c'	-1
'd'	4

Get the value associated with a key

```
my_dict['a'] # 1
```

```
my_dict['e'] # Raises KeyError: 'e'
```

get() method returns the value for a key if it exists.

If the key is not found, get() returns None (not an error)

```
my_dict.get('c') # -1
```

Writing to a Dictionary

Change the value or add a new key-value pair

```
my_dict['e'] = 5
```

Updated dictionary: {'a':1, 'b':17, 'c':-1, 'd':4, 'e':5}

Extending a dictionary with another dictionary

```
more_data= {'t': [0, 2, 4], 'k': "cheese"}
```

```
my_dict.update(more_data)
```

{'a': 1, 'b': 17, 'c': -1, 'd': 4, 'e': 5,

't': [0, 2, 4], 'k': 'cheese'}

Examples of Dictionary Usage

Grouping and Counting Items

Count how many times each word appears in a given string. To split a string into words separated by spaces and return them in a list, use the [`split\(\)`](#) method.

```
text = "apple banana apple orange banana apple"
word_count = {}
for word in text.split():
    if word in word_count:
        word_count[word] += 1
    else:
        word_count[word] = 1
print("Word count:", word_count)
# Word count: {'apple': 3, 'banana': 2, 'orange': 1}
```

Mapping a Key to Multiple Values

Given an input list of words, group words by their length.

```
words = ["cat", "dog", "elephant", "mouse", "rat", "lion"]
length_groups_dict = {}
for word in words:
    length = len(word)
    if length not in length_groups_dict:
        length_groups_dict[length] = [] # create an empty list
    length_groups_dict[length].append(word)
print("Words grouped by length:", length_groups_dict)
# Words grouped by length: {3:['cat', 'dog', 'rat'],
#                           8:['elephant'], 5:['mouse'], 4:['lion']}
```

Working with Dictionaries

Continued...

Creating a Dictionary with fromkeys()

- The `fromkeys()` method creates and returns a dictionary with the specified keys and the specified value (same for all keys)
- Usage:

`dict.fromkeys(keys, value)`

- `keys` is an iterable object (`list`, `string`, `tuple`, `range`), specifying the keys of the new dictionary
- `value` (optional) is the value for all keys
 - Default value is `None`

Creating a Dictionary with fromkeys()

Create a dictionary with (a) keys being integers from 0 to 3 and (b) all associated values empty lists.

```
new_dict = dict.fromkeys(range(4), [])
```

```
print("New dictionary with empty lists as values:")
```

```
print(new_dict)
```

```
New dictionary with empty lists as values:
```

```
{0: [], 1: [], 2: [], 3: []}
```

Creating a Dictionary with dict()

Create a dictionary using dict() from a list of tuples

```
students_list = [('Victoria', 286734),  
                 ('Jeremy', 234809),  
                 ('Katie', 256789)]
```

```
students = dict(students_list)
```

```
{'Victoria': 286734, 'Jeremy': 234809, 'Katie': 256789}
```

Extracting Keys or Values to a List

```
students = dict(students_list)
{'Victoria': 286734, 'Jeremy': 234809, 'Katie': 256789}
```

```
list(students.keys()) # List of all keys
# ['Victoria', 'Jeremy', 'Katie']
```

```
list(students.values()) # List of all values
# [286734, 234809, 256789]
```

```
list(students.items()) # List of key-value pairs as tuples
# [('Victoria', 286734), ('Jeremy', 234809), ('Katie', 256789)]
```

Looping Through Dictionary using items()

```
students = dict(students_list)
{'Victoria': 286734, 'Jeremy': 234809, 'Katie': 256789}

for name, sciper in students.items():
    print(f"Sciper number of {name} is {sciper}")

# Sciper number of Victoria is 286734
# Sciper number of Jeremy is 234809
# Sciper number of Katie is 256789
```

Sorting a Dictionary By Key

- By default, if sorted, dictionaries are sorted **by keys in ascending order** (alphabetically)

```
# A dictionary of items with their prices
```

```
fruits = {"date": 2.80, "banana": 1.20, "apple": 2.50,  
          "elderberry": 1.50, "cherry": 3.00}
```

```
# Sorting the dictionary by keys in ascending order
```

```
sorted_by_keys = dict(sorted(fruits.items()))
```

```
print(sorted_by_keys)
```

```
# {'apple': 2.5, 'banana': 1.2, 'cherry': 3.0,  
#  'date': 2.8, 'elderberry': 1.5}
```

Removing Dictionary Items

```
# students = {'Victoria':286734,'Jeremy':234809,'Katie':256789}

# pop() removes an entry, if present, and returns the value
# corresponding to the removed key
students.pop('Lena')          # KeyError: 'Lena'
students.pop('Victoria')     # 286734
# Updated dictionary: {'Jeremy': 234809, 'Katie': 256789}

# popitem() removes and returns the last inserted key-value pair.
# If the dictionary is empty, it raises an error KeyError
my_dict.popitem()            # ('Katie', 256789)
# Updated dictionary: {'Jeremy': 234809}
```

Removing Dictionary Items

```
# students = {'Victoria':286734, 'Jeremy':234809, 'Katie':256789}
```

```
# del keyword is used to delete an object (a variable of any  
# type); it can also be used to delete a key-value pair;
```

```
# if the key does not exist, it raises a KeyError
```

```
del students['Katie']
```

```
# removes key-value pair where 'Katie' is the key
```

```
# {'Victoria': 286734, 'Jeremy': 234809}
```

```
# clear() method empties a dictionary
```

```
my_dict.clear()
```

```
# {}
```



Next topic: Exam-style Question Examples