Basic Programming

Lesson 8. Collections: Dictionaries. JSON construction

Alexandra Ciobotaru

3 dec 2021

Syllabus

- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- **Lesson 4. Strings. Working with strings.**
- **Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While**
- **Lesson 6**. ~~Working with matrices in Python. Python numpy library~~**. Lesson 5 continued** Quiz 1 (25%).
- **Lesson 7. Creating functions. Recursive functions. Matrices**.
- **Lesson 8. Collections: Dictionaries. JSON construction.**
- Lesson 9. Working with files. Reading and Writing.
- Lesson 10. Analyzing dataframes. Pandas and Matplotlib Python libraries.
- Lesson 12. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism. Quiz 2 (25%).
- Lesson 13. Object-Oriented Programming (cont.). Error handling. Best practices when programming.
- Lab (20%) + final exam (30%).

What you'll learn today:

- 1. Sets
  - 1.1. What is a set?
  - 1.2. Accessing elements in a set
  - 1.3. Set methods
- 2. Dictionaries
  - 2.1. What is a dictionary?
  - 2.2. Accessing items of a dictionary
  - 2.3. Adding items to a dictionary
  - 2.4. Removing items from a dictionary
  - 2.5. Dictionary methods
  - 2.6. Looping through dictionaries
- 3. JSON construction

1. Sets
1.1. What is a set?

- A set is a collection of distinct objects, typically called **elements** or **members**.
- Just like lists and tuples, sets are used to store multiple items in a single variable.
- A set is a collection which is unordered, unchangeable and unindexed.
- Sets are written with curly brackets:

```
my_set = {‘apples’, ’bananas’, ’pears’}
print(my_set)
```

- Set items are **unordered**, **unchangeable**, and **do not allow duplicate values.**

```
>>> set1 = {"apple", "banana", "cherry", "apple"}
>>> print(set1)
set(['cherry', 'apple', 'banana'])
```

- Set items can be of **any data type**:

```
set1 = {"abc", 34, True, 40, "male"}
>>> print(set1)
set([40, True, 34, 'abc', 'male'])
```

- **Casting** is done using the keyword “set”:

```
>>> thisset = set(("apple", "banana", "cherry")) #
note the double round-brackets
>>> print(thisset)
set(['cherry', 'apple', 'banana'])
```

1.2. Accessing elements in a set

- You cannot access items in a set by referring to an index or a key.
- What you can do is loop through the set items using a for loop:

```
>>> thisset = {"apple", "banana", "cherry"}
>>> for x in thisset:
    print(x)
cherry
apple
banana
```

- Or ask if a specific element is in the set:

```
>>> print("banana" in thisset)
True
```

1.3. Set methods

- Adding an item:

```
>>> thisset = {"apple", "banana", "cherry"}
>>> thisset.add("orange")
>>> print(thisset)
set(['orange', 'cherry', 'apple', 'banana'])
```

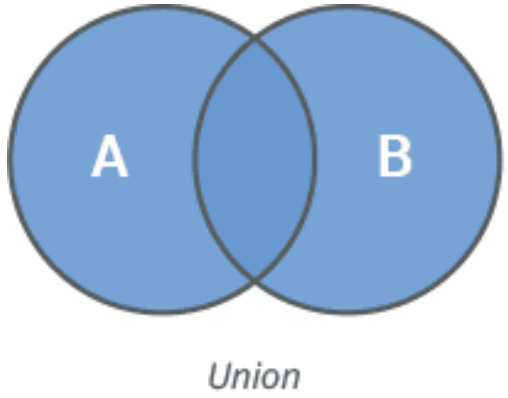
- Or add a set altogether:

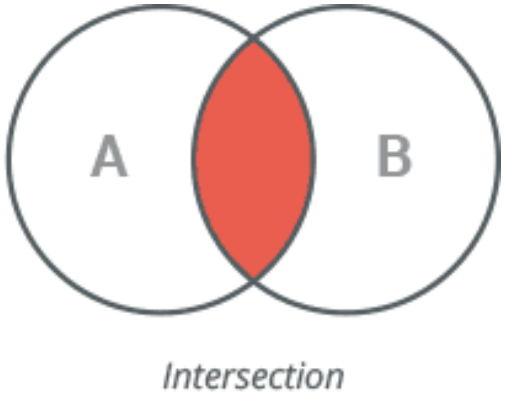
```
>>> thisset = {"apple", "banana", "cherry"}
>>> tropical = {"pineapple", "mango", "papaya"}
>>> thisset.update(tropical)
>>> print(thisset)
set(['mango', 'papaya', 'apple', 'pineapple', 'cherry',
'banana'])
```

- To remove an item in a set, use the **remove**(), or the **discard**() method in the same way.

Sets are commonly used for computing mathematical operations such as union, intersection, difference, and symmetric difference.

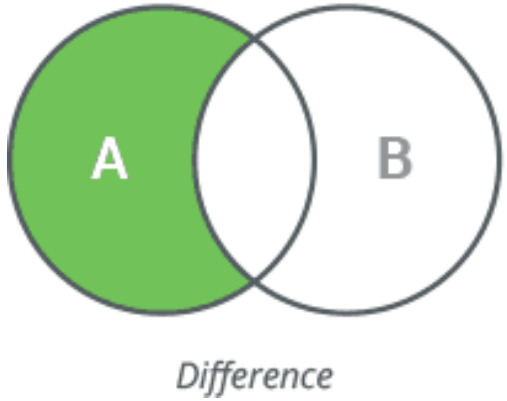
A.union(B)

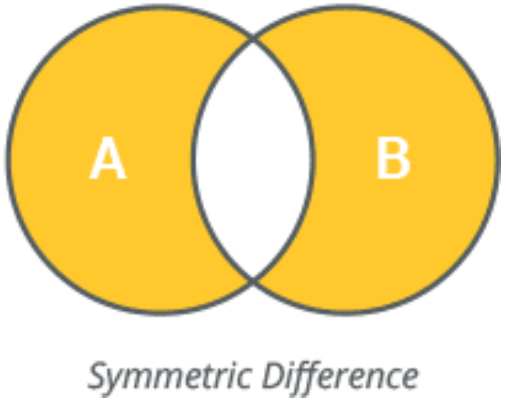




A.intersection(B)

A.difference(B)





A.symmetric_difference(B)

Image source: https://www.learnbyexample.org/python-set/

Also, sets are commonly used to automatically remove duplicates from a list.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

2. Dictionaries
2.1. What is a dictionary?

- Dictionaries are ***ordered****, ***changeable*** and ***not allowing duplicates*** collections of values, used to store data in the form of "***key – value***" pairs.

```
>>> thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> print(thisdict)
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- Creating an empty dictionary:

```
>>> empty_dict = {}
>>> print(empty_dict)
{}
```

* for Python 3.7 versions and up

Creating a dictionary from a **zip**

- The zip function combines two lists:

```
>>> number_list = [1, 2, 3]
>>> str_list = ['one','two','three']
>>> result = zip(number_list, str_list)
>>> print(result)
<zip object at 0x000002FB61F12A00>
>>> print(list(result))
[(2, 'two'), (1, 'one'), (3, 'three')]
```

I cold have used set instead of list
(useful when I want to eliminate duplicates)

- An iterable of this form can be transformed into a dictionary using the dict() function:

```
>>> print(dict(result))
{1: 'one', 2: 'two', 3: 'three'}
```

2.2. Accessing dictionary items

- You can access the items (the values) of a dictionary by referring to its key name, inside square brackets:

```
>>> thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> print(thisdict["model"])
Mustang
```

- There is also the method get() that does the same thing:

```
>>> print(thisdict.get("model"))
Mustang
```

- The keys() method will return a list of all the keys in the dictionary:

```
>>> print(thisdict.keys())
dict_keys(['brand', 'model', 'year'])
```

2.3. Adding items to the dictionary

- You can add a key-value pair to the dictionary by using the following formula: <name_of_dictionary>[<“key_name”>] = <key_value>

```
>>> car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
>>> car["color"] = "white"
>>> print(car)
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964,
'color': 'white'}
```

- Or by using the update() method:

```
>>> car.update({"doors":5})
>>> print(car)
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964,
'color': 'white', 'doors': 5}
```

2.4. Removing items from the dictionary

- The pop() method removes the item with the specified key name (popitem()* method removes the last inserted item):

```
>>> thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> thisdict.pop("model")
'Mustang'
>>> print(thisdict)
{'brand': 'Ford', 'year': 1964}
```

- Or by using del keyword:

```
>>> del thisdict["model"]
>>> print(thisdict)
{'brand': 'Ford', 'year': 1964}
```

The clear() method empties the dictionary:

```
>>> thisdict.clear()
>>> print(thisdict)
{}
```

* for Python 3.7 versions and up

2.5. Dictionary methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

2.6. Looping through dictionaries

- Looping a dictionary will by default return the *keys* of the dictionary:

```
>>> thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> for x in thisdict:
  print(x)
brand
model
year
```

```
for x in thisdict.keys():
  print(x)
```
(does the same thing)

- To return the *values*:

```
for x in thisdict:
   print(thisdict[x])
```

- Loop through both *keys* and *values*, by using the items() method:

```
for x, y in thisdict.items():
  print(x, y)
```

3. JSON construction

- JSON stand for JavaScript Object Notation
- It is a text format that is language independent (can be used in Python and other programming languages as well)
- The python native library that works with json: import **json.**
- Reading and writing a JSON file is much faster than reading and writing csv files or text files, hence its utility.

JSON example

```
{
  "title": "cognitive science rocks",
  "number of subjects": 5,
  "rocks": true,
  "subjects": [
    "Statistics",
    "Programming",
    "Artificial Intelligence"
  ],
  "my grades": {
    "Statistics": 10,
    "Programming": 10,
    "Artificial Intelligence": null
  }
}
```

A JSON file starts and ends with curly brackets and can contain the following datatypes (in the form of key-value pairs)

- string,
- number,
- Boolean,
- list,
- dictionary object.

Any value can be empty using the keyword ‘null’ (in this example, artificial intelligence hasn’t been graded yet).

Thank you!