Sets

Prof.Dr. Bahadır AKTUĞ
803400815021 Machine Learning with Python

**Compiled from sources given in the references*

Sets

- Similar to lists, sets can contain elements of different data types (strings / integers etc.)
- On the other hand, the elements of sets has to be immutable
- While the elements of a sets have to be immutable, the set itself is mutable
- Sets can be expanded or shrunk upon need
- One thing which distinguishes sets from other data types is that one element could take place in a set only once
- The sets in Python are very similar to the sets in Mathematics and the traditional set operations (subset, union, intersection, difference etc.) over Python sets are available and highly practical.

Sets

Defining a new set:

Set can be defined directly set() function or {} operator.

```
>>> cities = set(["Ankara", "Adana", "Samsun"])
>>> print(cities)
{'Ankara', 'Adana', 'Samsun'}
>>> cities = set({"Ankara":"06", "Adana": "01", "Samsun":"55"})
>>> print(cities)
{'Adana', 'Ankara', 'Samsun'}
>>> cities = {"Ankara", "Adana", "Samsun"}
>>> print(cities)
{'Adana', 'Ankara', 'Samsun'}
>>> x = set("EEE105")
>>> print(x)
{'E', '0', '5', '1'}
```

Set Functions

add():

The elements of sets have to be immutable. Thus, the argument for the "add()" function has to be immutable.

```
>>> colors = set(["Yellow", "Blue", "Green"])
>>> colors.add('Red')
>>> print(colors)
{'Green', 'Red', 'Blue', 'Yellow'}
>>> colors.add (['Red','Orange'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Set Functions

update():

An element can be added to an existing set by using add() function. If more than one elements are to be added, the "update()" function can be used.

```
>>> colors = set(["Yellow","Blue","Green"])
>>> colors.update(["Orange","Red"])
>>> print(colors)
{'Green', 'Red', 'Blue', 'Orange', 'Yellow'}
```

Set Functions

clear():

Deletes (clears) all the elements in a set. However, it does not delete the set itself, just makes it an empty set.

```
>>> colors = set(["Yellow","Blue","Green"])
>>> colors.clear()
>>> print(colors)
set()
```

Set Functions

copy():

As with other data types, direct assignment of sets (the are all objects) could lead to disastrous results. "copy()" function, as the name implies, copies the elements of a set to another set.

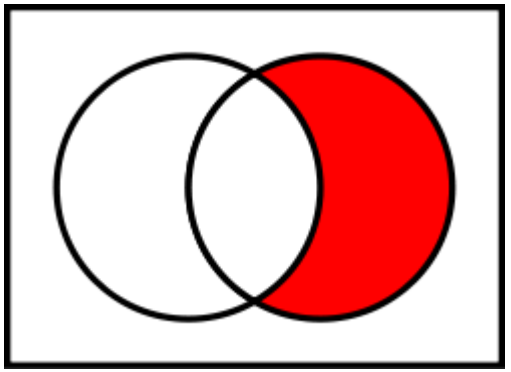
```
>>> colors = set(["Yellow", "Blue", "Green"])
>>> colors2 = colors
>>> colors2.clear()
>>> print(colors)
set()

>>> colors = set(["Yellow","Blue","Green"])
>>> colors2 = colors.copy()
>>> colors2.clear()
>>> print(colors)
{'Green', 'Blue', 'Yellow'}
```

Set Functions

difference(): (-)

Gives the difference between a set and another set (A \ B).



```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> z = {"c","d"}
>>> x.difference(y)
{'a', 'e', 'd'}
>>> x.difference(y).difference(z) {'a', 'e'}
```

The same operation can be done with "-" operator:

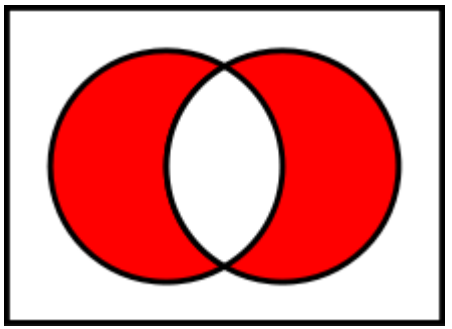
```
>>> x - y {'a', 'e', 'd'}
>>> x - y – z
{'a', 'e'}
```

Set Functions

symmetric_difference(): (^)

Corresponds to **disjunctive union** and gives the difference (non-overlapping parts) for both sets.

```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> z = {"c","d"}
>>> x ^ y
{'a', 'd', 'e'}
>>> y ^ z
{'b', 'd'}
>>> y - z
{'b'}
```



Set Functions

difference_update():

This is a combined operation of the difference (A \ B) and updating the original set (A) with the difference (A = A \ B)

```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> x.difference_update(y)
>>> print(x)
{'d', 'a', 'e'}
```

Set Functions

discard(element):

Deletes an element from a set. If the element is not in the set, it does not return an error message.

```
>>> x = {"a","b","c","d","e"}
>>> x.discard("a")
>>> x {'c', 'b', 'e', 'd'}
>>> x.discard("z")
>>> x
{'c', 'b', 'e', 'd'}
```

Set Functions

remove(element):

While similar to "discard" function, it gives an error message if the element to be removed is not in the set.

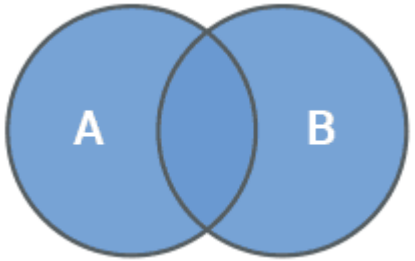
```
>>> x = {"a","b","c","d","e"}
>>> x.remove("a")
>>> x
{'b', 'c', 'd', 'e'}
>>> x.remove("z")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'z'
```

Set Functions

union(set(s)): (|)

Return the union of a set with one or more sets.

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x.union(y)
{'a', 'd', 'g', 'b', 'c', 'e', 'f'}
```



OR

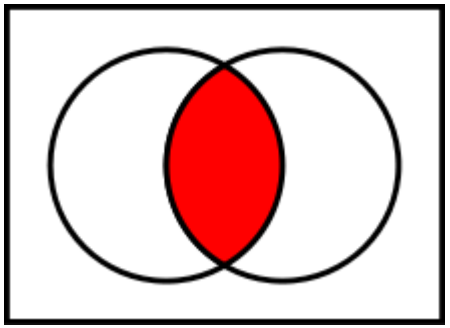
```
>>> x | y
{'a', 'd', 'g', 'b', 'c', 'e', 'f'}
```

Set Functions

intersection(set): (&)

Return the intersection of a set with another set.

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x.intersection(y)
{'c', 'e', 'd'}
```



OR

```
>>> x & y
{'c', 'e', 'd'}
```

Set Functions

isdisjoint()

Returns a boolean expression about whether a set is disjoint with another set.

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d","e","f","g"}
>>> x.isdisjoint(y)
False

>>> x = {"a","b","c","d","e"}
>>> y = {"h","j","k","f","g"}
>>> x.isdisjoint(y)
True
```

Set Functions

issubset() (<=)

Returns a boolean expression about whether a set is a subset of another set.

```
>>> x = {"c","d"}
>>> y = {"a","b","c","d","e"}
>>> x.issubset(y)
True

>>> x = {"c","d"}
>>> y = {"a","b","c","d","e"}
>>> x <= y
True
```

Set Functions

issuperset() (>=)

Returns a boolean expression about whether a set is a superset of another set.

```
>>> x = {"c","d"}
>>> y = {"a","b","c","d","e"}
>>> y.issuperset(x)
True

>>> x = {"c","d"}
>>> y = {"a","b","c","d","e"}
>>> y >= x
True
```

Set Functions

pop()

This function returns a random element of the set and deletes that element from the set.

```
>>> x = {"a","b","c","d","e"}
>>> x.pop()
'b'
>>> print(x)
{'c', 'a', 'd', 'e'}
>>> x.pop()
'c'
>>> print(x)
{'a', 'd', 'e'}
```

Set Functions

Multiple Sets:

Union and intersection functions can operate on more than one sets.

```
>>> x = {"a","b","c","d","e"}
>>> y = {"h","j","c","d","k"}
>>> z = {"i","b","e","d","n"}
>>> set.intersection(x,y,z)
{'d'}
>>> set.union(x,y,z)
{'j', 'd', 'i', 'b', 'c', 'a', 'k', 'n', 'h', 'e'}
```

Set Functions

Checking if element of a set:

```
>>> x = {"a","b","c","d","e"}
>>> 'c' in x
True
>>> 'p' in x
False
```

Set Functions

Loops over set elements:

```
>>> x = {"a","b","c","d","e"}
>>> for harf in x:
...   print(harf)
...
b
c
a
d
e
```

Set Functions

Frozen Sets:

While "Frozen sets" have the same properties as the normal sets, as the name implies, they cannot be modified once defined. Such immutable sets have further usage. For instance, they can keys for dictionaries and they can elements can other sets.

```
>>> x = frozenset(["a","b","c","d","e"])
>>> y = set([x,3,4,5])
>>> y
{3, 4, frozenset({'b', 'c', 'a', 'd', 'e'}), 5}
>>> x.add('f')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

Input & Output

Prof.Dr. Bahadır AKTUĞ

803400815021 Machine Learning with Python

"Reading from"/"Writing to" Console

- Reading from console is done with "input" command since Python version 3.x.
- For older versions (e.g. 2.7 etc.), the command for reading from console is "raw_input()".
- General form of the command;

```
>>> a = input()
```

- Only one datatype can be read with "input" command: string. If further arithmetic operations are needed, the string read with "input" has to be transformed taking the type of the data into account (integer, float etc.).
- To write to console, "print" command is used. "print" is a function since Python version 3.0.

Reading from file

- File operations in Python are similar to many other high level languages.
- To read from a file, a file pointer needs to assigned which points to the physical file on disk and the mode of operation (read, write, append etc.

```
>>> fid = open("ogrenci_listesi.txt", "r")
```

- 'r' stands for the reading mode of the file operation.
- The default mode is "reading" if no mode is given. However, a good practice of programming is to indicate the mode of operation explicitly.

Reading from file

- After opening the file (having an identifer for the file), The following commands can be used to read from the file:
  - read(byte-sayısı)
  - readline()
  - readlines()
- a = read(n) command reads n byte data from the file and assigns it to the variable "a".
- a = readline() command reads one line of code from the file and assigns it to the variable "a".
- a = readlines() command reads all the lines in the file and assigns it to the variable "a".

Reading from file

- File opened with "open()" command should be closed with "close()" command.
- To close a file the file identifer is needed;

```
>>> fid.close()
```

- Another useful command is the "with/as" command.
- If the file is opened with "with/as" command, then the file is automatically closed at the end.

```
with open('output.txt', 'w') as f:
    f.write('Hi there!')
```

File opening modes

Mode	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Writing to file

- The command to write to a file is "write() and is used with the file identifier;

```
>>> fid.write()
```

- The "write" command can have the following parameters;
  - writing format
  - the variables to write to the file
- The writing format contains the data type of the variable to be written, number of digits after the decimal point, total number of digits allocated for each variable.

Formatted Output

```
>>> write("Number:  %5d, Price: %8.2f" % (12,54.70))
```

- Here
  - %5d, denotes that an integer with 5 digits will be used
  - %8.2f, denotes that a float number with a total number of 8 digits (including decimal point and sign) will be used and 2 of the digits will be after the decimal point.
- The variable list has to be a tuple.
- "%" operator should not be missed in the format string.

Reading all the lines in a file

- If the complete file will be read, for or while loops can be used:
- with "while";

```
line = fid.readline()
while line:
    line = fid.readline()
     print(line)
```

- with "for";

```
for line in fid.readlines():
    print(line)
```

Format string commands

`d`	Signed integer decimal.
`i`	Signed integer decimal.
`o`	Unsigned octal.
`u`	Unsigned decimal.
`x`	Unsigned hexadecimal (lowercase).
`X`	Unsigned hexadecimal (uppercase).
`e`	Floating point exponential format (lowercase).
`E`	Floating point exponential format (uppercase).
`f`	Floating point decimal format.
`F`	Floating point decimal format.
`g`	Same as "`e`" if exponent is greater than -4 or less than precision, "`f`" otherwise.
`G`	Same as "`E`" if exponent is greater than -4 or less than precision, "`F`" otherwise.
`c`	Single character (accepts integer or single character string).
`r`	String (converts any python object using `repr()`).
`s`	String (converts any python object using `str()`).
`%`	No argument is converted, results in a "`%`" character in the result.

Examples

```
>>> print("%10.3e"% (356.08977))
3.561e+02
>>> print("%10.3E"% (356.08977))
3.561E+02
>>> print("%10o"% (25))
31
>>> print("%10.3o"% (25))
031
>>> print("%10.5o"% (25))
00031
>>> print("%5x"% (47))
2f
>>> print("%5.4x"% (47))
002f
>>> print("%5.4X"% (47))
002F
>>> print("Only one percentage sign: %% " % ())
%
```

Examples

```
>>> print("%#5X"% (47))
0X2F
>>> print("%5X"% (47))
2F
>>> print("%#5.4X"% (47))
0X002F
>>> print("%#5o"% (25))
0o31
>>> print("%+d"% (42))
+42
>>> print("% d"% (42))
42
>>> print("%+2d"% (42))
+42
>>> print("% 2d"% (42))
42
>>> print("%2d"% (42))
42
```

Formatting string variables

- Another method is the preparation of string with "format" command before writing to the screen or to the file.
- For this purpose, the string type variable has "{}" to denote the formatting of the variables, constants within the string.

```
>>> "Number:  {0:5d}, Unit Price: {1:8.2f} ".format(12,54.70)
```

- Here
  - {0} stands for the first argument, 5d stands for integer formatting with 5 digits.
  - {1} stands for the second argument, 8.2f stands for float formatting with a total of 8 digits and with 2 of them are after the decimal point.
- Formatted string can be written as usual.

References

1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). How to Think Like a Computer Scientist: Learning with Python (3nd edition).
2 Pilgrim, M. (2014). Dive into Python 3 by. Free online version: DiveIntoPython3.org ISBN: 978-1430224150.
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
5 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
6 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
7 John D. Hunter (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95.
8 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
10 http://www.diveintopython.net/
11 https://docs.python.org/3/tutorial/
12 http://www.python-course.eu
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/